Mecrisp Stellaris Unofficial UserDoc

THINKING CAP

- Index
- WHAT’S NEW ?
- Supported Hardware
- MCU Documentation
- Released under Version 3 of the GPL, Mecrisp-Stellaris Forth by Matthias Koch is Free Software.
    - See https://sourceforge.net/projects/mecrisp/files/mecrisp-stellaris-2.5.3.tar.gz to download the latest version for Cortex-M processors.
- Contact Site Maintainer

This Table Shows Why CMSIS Matters.

MCU	Peripherals	Registers	Register Bitfields

MCU	Peripherals	Registers	Register Bitfields
MSP430G2553	17	113	503
STM32F0xx	37	413	3044
STM32F303x	38	549	3857
STM32F103xx	53	722	4833
GD32F103	?	?	? (I need a svd)
STM32F103C8	33	466	3503
GD32VF103Cx	46	1074	7713
TM4C129x	65	2137	7347
STM32F7x7	93	2093	17051

Contents:

- Quickstart
    - FORTH LOVE? IF HONK THEN
    - What is Mecrisp-Stellaris ?

    - What is Forth ?
    - Quick Start Here
    - Where to go now ?
    - Where to go next ?
    - Any other tips ?
    - I’ve found some bugs, horrible spelling errors, or have some good tips, corrections etc, who do I contact ?
- Common User Issues
    - Lockups
    - GPIO Manipulation
    - How To Turnkey
    - Different Forths Have Different Words!
    - My Solution Isn’t Here!!
- Dictionary
    - Example Dictionary Listing
    - Forth Definition Components
    - RA Core Register Usage
    - Number Bases
    - Data Types
    - Key to Tables
    - Compiler Error Messages
    - Terminal-IO
    - Stack Jugglers
    - Logic
    - Single Number Math
    - Double Number Math
    - Fixed Point Arithmetic
    - Decision Making
    - Bits: Test, Toggle, Clear and Set
    - Memory Status
    - Memory Access
    - String Formatting
    - Deep Insights
    - User Input and Interpretation
    - Dictionary Expansion

  - Flags and Inventory
  - Folding
  - Compiler Essentials
  - Control Structures
  - Common Hardware Access
- About This Site
  - About the Author
  - Contact The Author
  - Documentation Copyright
  - Program Copyrights
- DOCUMENTATION FOR MCU(s)
  - Example
- Alternate Functions
  - GPIOX_MODER (STM32F051)
  - GPIOA Alternate Function Matrix (STM32F051)
  - GPIOB Alternate Function Matrix (STM32F051)
  - GPIO Alternate Function Selection Codes (STM32F051)
  - AFR Example
- Antiforth
  - ANSI Forth is ANTI Forth
- Arrays
  - Data Types
  - Emulate c,
  - Create Array of Halfwords
  - Create Array of Cells
- Asyncronous Message Passing
  - Article from Comp Language Forth (Usenet)
- Buffers
  - Buffer Creation (Bytes)
  - Buffers: Storing Data
  - Buffers: Fetching Data
  - Practical Usage
  - Buffer Creation (Cells)
  - The Buffer XT
  - Show Packed Data

  - Dump the Buffer
  - Proving a buffer only holds X bytes
- Assembly Language: Interactive
  - Environment
  - Software Tools Required
  - Registers
  - The Forth Blinky
  - Assembly Language ‘asm-led-off’ Word
  - Assembly Language ‘asm-delay’ Word
  - Summary
  - Assembler-m0.txt Opcode Tables
- Autobaud
  - Autobaud for a Stm32F0xx Kernel
- Atomic read/modify accesses
  - STM32
- Beginner Advice
  - All Forths Are Different
  - No Halting on Errors.
  - Lockups
  - Classic Beginner Mistake
  - Why won’t my programs work ?
  - Stack problems
  - Long Words
- Benchmarks
  - Programming Language Benchmark
  - Power Of A Language Or Performace Of An Implementation ?
  - Benchmarking Different Forths/Mcus
  - Code
- BITFIELDS AND STRUCTURES
- BITFIELDS, DECLARE EARLY ?
  - Why I Use and Declare Bitfields Early
  - Electronics Technician Style
  - Summary
- Bit Manipulation Tips
  - XOR Bit Resetting

    - GPIOx_MODER Table
- Bugs or Benefits ?
    - Unhandled interrupt 0000000X !
    - Old Words Remain After Upgrading Mecrisp-Stellaris!
    - Unhandled interrupt 00000003
- Bluepill Examples
    - Traffic Lights
- C and Forth Examples
- Charles (Chuck) Moore the inventor of Forth
- Chips
    - This Months HOT MCU
    - A Most Beautiful Chip, the STM32L162RD
    - Block Diagram
    - Capability
    - The Perfect Chip Right?
    - Runs Mecrisp-Stellaris?
- Cloning, Manual
    - iHex
- Cloning, Integrated
    - Cloning, A New Method
    - Dependencies
    - Operation
    - Downloads
    - clone.sh
- Code Errors, Detecting
    - Sourcecode Errors
    - Kernel Mod Features
    - Running Picocom
    - F0 Disco Hardware Handshaking Connections
    - Schematic Diagram
    - Kernel
- Code Reviews and Libraries
    - Can Forth Code be Reviewed, are Libraries Useful ?
    - Most Forth Development Environments Are Different
- Colour
    - Words to Provide Coloured Characters

- Common Programs
    - assembler-m0.txt
    - bitlog.txt
    - blocks.txt
    - calltrace.txt
    - charcomma.txt
    - conditional.txt
    - disassembler-m0.txt
    - disassembler-m3.txt
    - dump.txt
    - editor.txt
    - examples.txt
    - experimental/
    - floored-divide.txt
    - fonts/
    - Flash
    - graphics-unicode-8x16.txt
    - graphics-unicode-8x8.txt
    - graphics.txt
    - longcomment.txt
    - mecrisp-registergenerator/
    - multitask.txt
    - nvic.txt
    - profiler.txt
    - prompt.txt
    - pseudorandom.txt
    - quotations.txt
    - registergenerator/
    - romans.txt
    - sine.txt
    - sqrt.txt
    - svd2forth-v2/
    - Exceptions
- Contributing Code
- Contributors Go Here
- Critical Systems

  - Toyota
  - Safety Research and Strategies Inc
  - BOOKOUT V. TOYOTA
- Data Stack
  - A Succint Stack Description by Dave Wyland
- Defining Words
  - virginia.edu
  - forth.org
  - The basics of create … does>
  - An example
  - Stage 1: Compiling the defining word
  - Stage 2: Creating a “child”
  - Stage 3: Executing the child
  - Simple Example with LOOP
- Delay
  - Accurate Blocking Delay
  - Why do it this way ?
  - Using Pure Forth Instead
  - Delay Measurments
  - The Method below produces the following delays:
  - Delay Code
  - Delay Measurments
  - Millisecond Delay
  - Example usage
  - How to Inline code
- FORTH DEVELOPMENT ENVIRONMENT
  - Trqaditional Forth
  - My Development System
  - Overall Operation
  - PROJECT BUILDER
  - SERIAL TERMINAL
  - CrystalScad
- Development Tips
  - PR: Monitor Stack Values and Program Flow
  - Balancing The Return Stack
  - Compare Local Variable vs Return Stack code size

- Emacs and Forth
  - To Do:
- Endianess
  - Little-endian vs. big-endian
  - Doing math vs. indexing/sorting/searching
  - Byte order vs. bit order
  - Memory access
- Example Forth Programs
- Examples: Word Usage
  - WFI ($BF30)
  - <builds does>
  - bit@
  - compare
  - +loop
  - loop
  - min
  - max
  - min-max window
  - umin
  - Return Stack
  - Roll, -Roll
- File System
  - Mecrisp-Stellaris Block Implementation
  - Block Storage Hardware
  - SDcard
- F0 Discovery: 48 MHz Clock
  - Sourcecode
- F0 Discovery Blinkies
  - What *is* a blinky ?
  - Blinky 1
  - Blinky 2
  - Blinky 3
  - Blinky 4
  - Blinky 5
  - Blinky 6
  - Blinky 7

  - Blinky 8
  - Exercises For the Reader
- STM32F1 USB
  - Download
  - EEVblog Forum Feedback
  - STM32ID
  - NON STM32F103 MCU’s For Comparison
  - STM32F103 or Clone Results
  - 1882bdaff62f637a7bdd517d14f4c1a4.bin User Results
- GD32VF103 RISC-V
  - Features
  - Interesting Features
  - Mecrisp-Quintus
  - Premade Boards
  - Wio Lite RISC-V (GD32VF103) - With ESP8266
  - Polos GD32V Alef
  - Connect a Terminal
  - Utility Programs Ported So Far
  - Compiling
  - Downloads
- Fixed Point (s31.32)
  - Examples
  - s31.32 Insights
- The First Rule of Forth
  - The Polarizing Effect
- Flashing Mecrisp-Stellaris onto a STM32: Methods
  - Flashing Methods
  - Serial Bootloader
  - Saving a on chip binary
  - Stlink
  - DF-UTIL
  - SWD
  - JTAG
- Forth Development Applications
  - Asciio

  - Graphviz
- Folding
  - x-foldable
  - Folding Example
- Forth (embedded) Interactive Alternatives for Cortex M on STM32F
  - Micropython
  - How does Micropython Work?
  - Testing Micropython
  - eLua
  - Armpit Scheme
  - Summary
- Forth Books
  - Modern Forth Books
  - The two must read *classic* books
  - Forth Book Lists
- Forth Links
  - Mecrisp-stellaris
  - Why I Use Forth by Bernd Paysan
  - Forth Still Suits Embedded Applications (Tom Napier | Nov 21, 1999)
  - The Forth Programming Language - Why YOU should learn it
  - Forth - the LEGO of Programming Languages 1995
  - Why Forth ?
  - Old Tutorial List
  - Forths at Github
  - Space-Related Applications of Forth
- Forth Variants
  - Direct threaded code.
  - Subroutine-threaded code
- Forth Versus Other Systems
  - Riyadh, Saudi Arabia Airport Project
- Forth Based Commercial Products
  - Ground Station Dish Controller
- Mecrisp-Stellaris Videos

  - Josh shows some of the fun he has been having with the Mecrisp-Stellaris interactive programming environment
- Forth Wizard
  - A Stack Manipulation Tool
- Fossil Distributed Version Control System
  - Fossil Quickstart Page:
  - An excellent Fossil Summary page by Brandon Invergo
  - The Fossil Bbook
- Garbage Collection
  - Forth Does Not Need Garbage Collection
- Glossary
  - Cornerstone
  - Imm
  - <builds does>
  - Folding
  - Inlining
  - Smudge
  - Vocabularies
  - lshift
  - Loop Unrolling
  - Postpone
  - Registerliteral,
- HTML
- Low Power on STM32F0
  - Documentation
  - STM32F0 Low Power
  - My Example Forth Code
- Ihex 32 Bit
  - Forth Project Cloning
  - iHex.fs
  - Intel Hex
  - Ihex 04 Type Extended Addresses
  - Ihex 04 Type Extended Addresses Example
  - Ihex Dump Example

  - Description
  - Operation
  - Sourcecode
- Loadmeter
  - Cooperative Multitasking
  - How Does It Work ?
  - Calibration
  - Usage
  - Schematic Diagram
  - Source Code and Notes
  - Loadmeter Pics
- Interactive Assembly Delay
  - Why do it this way ?
  - Using Pure Forth Instead
  - Interactive Assembly
  - How does it work ?
  - Assembly Examples
  - Disassembly
  - Usage
- Interpreter: Inner
- Interpreter: Outer
  - Quit
- Interrupts
  - Cortex M0 Interrupt Flowchart
  - NVIC
  - EXTI
  - TABLE A: GPIO EXTI Mapping
  - Hardware Interrupt Selection
  - Hardware event selection
  - Software interrupt/event selection
  - Interrupt Service Routine
  - Forth Interrupt Methodology
  - Interrupt Examples
  - Systick is special!
- Intro: Forth
  - When was Forth Created ?

  - Is Forth Interpreted or Compiled ?
  - The Forth Philosophy
  - Forth: strengths and weaknesses
  - Mecrisp-Stellaris Forth
- Jump To Far
  - Error Description
  - Example
- Kernel Deep Configurations
  - Kernel Config File Name?
  - Stack Size
- Levels of Code in Forth Programming
- Library
  - Maths Library
  - Read Cortex M0 chip ID
  - 40 word dictionary pager
  - Stack Check
  - Print Register Contents with Legends
- GNU General Public License
- Words4
  - Viewing The Mecrisp-Stellaris Dictionary
  - Words
  - List
  - Words4
  - Words4 Creation
  - Words4 Source
  - Words4 Operation
  - Coloured Columns
- Machine and Assembly Code
  - Ra Register Allocator Usage
  - Executing Assembly/Machine Code
  - Notes
  - Example 1: 2plus (adds 2 to any number)
  - Example 2, more complex code, A blinky
  - Comparison to a Blinky written in Forth
- Mecrisp-Stellaris Changelog
- Mecrisp-Stellaris Known Issues

- Mecrisp-Stellaris Readme
- Memstat
  - STM32F103C8
  - GD32VF103
  - Code
- Mcu Specific Special Words
  - LM4F120
  - TM4C1294
  - MSP432P401R
  - LPC1114FN28
  - LPC1115FBD48
  - STM32F051
  - STM32F030F4
  - Interrupts
  - STM32L053C8
  - STM32L152
  - STM32L476
  - STM32F100
  - STM32F103
  - STM32F303
  - STM32F401
  - STM32F407
  - STM32F411
  - STM32F429
  - KL25Z128
  - XMC1100
  - EFM32GG990
  - MK20DX256
  - nRF51822
- Miscellanious
- A Modern Forth IDE
  - Youtube Videos
  - Historical Forth Development
  - Modern Development Systems
  - C v/s Historical Forth
  - C v/s Modern Forth

  - Modern Forth IDE Elements
  - Serial Terminal: GnuScreen
  - Speedup Methods Employed
  - Code editor
  - Bourne Shell
  - Coding/library support
  - Fossil SCM
  - Target Baud Rate increase to 480600
  - Hardware handshaking (RTS)
  - Automatic source-code error detection
  - Add CR to LF
- MSP340
  - Svd2Mecrisp
  - Sample Svd2Mecrisp output
- Nokia 5110 LCD Display
  - Description
  - Documentation
  - Synopsis
  - Research
  - Software Design
  - Hardware and Software Used
  - Pinouts and Signals
  - The very *First* Test
  - 5110 *Essential* Initialization
  - Logic Timing Diagram
  - All Pixels On Command (9)
  - Photo of all pixels on
  - Forth Code
- Numworks Calculator
  - Readme
- Patches: Kernel
  - Testing A Patch Before Applying
  - Applying patches
  - Reversing Patches
  - Create a Patch
  - Patch: Simpler Stack Printer

  - STM32F0
  - Rebuild Your Kernel
  - Compile example for mecrisp-stellaris-x.x.x
  - STM32F103 PATCH
  - Patching
- Porters Guide
  - 1) Find a close relative
  - 2) Install the Image from a similar chip onto your chip/board
  - 3) Connect a USB/3/3v Dongle
  - Garbled Text
  - 4) It’s running!
  - 5) Change Memory Values
  - 6) Changing MCU Name
  - 7) Building a new kernel
  - Test New Kernel
  - Checksum The Kernel
  - 8) Write a README
  - Flash Memory Operation
- Problem MCU’s
  - Atmel SAM based chips
  - Cypress PSoC’s
- Project: LMT01 Temperature Sensor
  - Example Output
  - YouTube Video
  - The LMT01 Temperature Sensor
  - Software Flowchart
  - The schematic
  - LMT01 Software
  - Downloads
- A Blue Pill LMT01 Temperature Sensor
  - USB
  - USART2
  - Screenshot
  - Instructions
  - Schematic

  - Blue Pill Layout
  - Tarball Contents
  - Source Code
  - Play With Forth ?
- Prototyping
  - Hardware
  - Examples
  - Copper Rivets
  - Source of suitable wire
  - A deadbugged STM32F051, hand soldered
- Radiation Hardened CPU
  - The RTX2010
  - Philae Lander
- Random Access Memory
  - Free Ram ?
- RA Kernel
  - RA Register Map
  - RA Bitexp Example
  - See bitexp
  - Convert your Classic Kernel to RA
  - Classic Mecrisp-Stellaris
  - Mecrisp-Stellaris Deeper Configuration
- Rants
  - I’m inspired by the epic rants of ttm richter
  - QUICK, GRAB A BARGAIN
  - A BARGAIN?
  - Summary
  - REINVENTING THE WHEEL
  - THE C PROGRAMMING LANGUAGE
  - THE C LIBRARY MYTH
  - USE MICROSOFT WINDOWS OR DIE
  - The Water Barons
  - THE HARD WAY
  - Flowchart
  - Keep It Simple
  - Asking For Help

  - A Learning Challenge
  - Hackerspaces or SalesPlaces?
  - Today’s Gold Standards
  - Tethered Forth
- Projects: Working Forth Programs
  - Window Comparator
  - A Sunrise and Sunset Calculator
  - Benchmark Your Forth MCU ?
  - Push button switch debounce demo
  - Square Wave Generator
  - 100 Microsecond Systick
  - Overclock to 96 MHz
  - F0 Discovery 48 MHz System Clock
  - Delay using the interrupt driven 1 millisecond Systick
  - Timer, 1mS Interrupt Systick
  - Analog To digital 19ch scanner
  - Morse code generator
- Recognisers
  - What is a recogniser ?
- References
- CMSIS-SVD
  - CMIS-SVD
  - Svd2forth
  - Svd2forth Downloads
  - Non STM32 svd2forth Downloads
  - Latest STM32 Release
- The Mecrisp REPL
  - Terminal handlers
  - Repl Maximum Character Length
  - Ledcomm Example from Matthias
- Serial Terminal: Software
  - GnuScreen
  - e4thcom-0.6.1
  - Picocom
- Source Documentation

    - In-Source Text : Configuration Code
    - In-Source Text : Program Code
    - Flowchart Source Code
    - Flowchart Picture
- Special Chip?
    - Where To start ?
    - What Language is Mecrisp Written in ?
    - Methods for adding a new MCU break down into two groups
    - Building Mecrisp-Stellaris
    - Sending Matthias Koch a Board
    - Hacking Mecrisp-Stellaris
    - My Mecrisp-Stellaris Hacks
- Stair-Stepping
    - The Solution ?
- Stacks
    - What is the default Stack size ?
    - Can Stack Size Be Changed?
- Stack Sensitive Prompt
    - A stack sensitive prompt
    - The code
- Stepper Motor Demo
    - Short Youtube Video
    - Hearing Impaired
    - Parts Used
    - Fourth Source Code
    - Schematic
    - Sophisticated Stepper Driver
- STM32 Book List
- STM32 Boards
    - STM32F Discovery Boards
    - Notes about the STM32F103C8T6 used in the “Blue Pill” and “Shenzhen-LC”
    - STM32F103 Limitations
    - Very Cheap Chinese STM32F103C8T6 Boards
    - Shenzhen-LC “mini STM32F103C8T6-board”

    - Why not make your own “GREEN PILL”
- STM32 Links
    - STM32 Wiki
    - STM32F Cortex M0 manual
    - Cortex Microcontroller Software Interface Standard (CMSIS)
    - GNU Binutils
- STM32F103C8 V/S GD32VF103Cx
    - Mecrisp Support
    - Specifications
    - Peripherals
    - ISA Comparisons
    - Premade Boards
- STM32F103C8T6, 2 x 64kB flash ?
    - STM32F103C8T6 #1 with 128kB Flash
    - STM32F103C8T6 #2 with 64kB Indicated Flash
    - Getting 128kB Flash!
    - stm32f1x.cfg Modification
    - Thanks “Bingo”!
- Storage Space
    - Using Unused Dictionary Space
- Supported Hardware
- Svd2gas - The Freedom To Assemble
    - Machine Code
    - Assembly Language
    - svd2gas
    - CMSIS
    - CMSIS-SVD
    - Assembler Toolchain
    - svd2gas README
    - Getting svd2gas
    - Demo Assembly Blinky
- Svd2forth-v3
    - Svd2forth-v3 Peripheral Register Pretty Printing
    - Svd2forth-v3 differs from V2 in the following ways
    - These notes

  - Howto
  - Printing a Peripheral
  - Printing a Register
  - Using bitfields.fs in your code
  - RCC Memmap.fs Example
  - RCC_CR2 bitfields.fs Example
  - RCC_CR2 Pretty Print Example
  - Svd2forth Downloads
- Serial Terminal: Hardware
  - USB-3.3v Dongle or cables
  - Serial Pin Connection Table For Various Boards
  - Note: Direct MCU USB access
- Serial Terminal: Hardware Flow Control
  - RTS/CTS flow control explained
  - Mecrisp-Stellaris Hardware used
  - First Things First, test your serial terminal
  - Handshaking Timing figure
  - STM32F0 discovery board mods for hardware flow control
  - CTS
  - RTS
- Thinking Forth Tips
- Thumb Instruction Set
- Thumbulator
  - Run Mecrisp-Stellaris-stm32f051-RA on your PC!
- Turnkey
  - Supported MCU’s
  - Thumbulator
- Programming Question And Answers
  - Editing Words
- VIM
  - Svd2forth
  - Bitfields.fs
  - Summary
  - Inserting a bitfield template into your code
  - Vim Completion

  - Vimrc
- Visualising the Forth Stack
  - Forth looks like a Stealth Aircraft
  - Disruptively Simple
  - The Stack
  - Minmax with Stack Diagrams
- wordlists, vocabularies, vocabulary prefixes and implicit context switching
  - Software Versions used in this demo
- XON/XOFF
  - Note from Matthias:
- YouTube Videos about Forth
  - Videos
  - My Videos

> What’s New ? <

Quickstart

> If you need help with any of this I can be found on IRC at #mecrisp-stellaris (irc.freenode). I’m in Australia, and usually available (Nick:tp) from 0900 to 2000 EST Australian Eastern Time.

FORTH LOVE? IF HONK THEN

This site assumes you are familiar with Forth and know how to use it.

If you’re a Forth beginner may I recommend the classic book, “Starting Forth” by Leo Brodie and “Thinking Forth”, by Leo Brody ?

Another great book is Stephen Pelc’s excellent MPE Forth, which is tailored for MPE Forth but has a lot of great Forth tips.

These three books are free, please see my books link here.

Mecrisp-Stellaris Forth YouTube Videos you may find interesting.

What is Mecrisp-Stellaris ?

- Released under Version 3 of the GPL, **Mecrisp-Stellaris Forth by Matthias Koch is Free Software**. **See:**

  https://sourceforge.net/projects/mecrisp/files/ **to download** your version
- Mecrisp-Stellaris Forth runs on M0, M3, M4 and M7 ARM Cortex M chips. This is a big family of microcontrollers with a common processor architecture and very different peripheral capabilities.
- The Mecrisp-Stellaris Distribution tarball comes with ready to run binaries for all supported chips, you don’t need to compile anything.
- Despite the name, not only Stellaris chips are supported.
- Mecrisp-Stellaris Forth fits into 19 kb of flash and runs with at least 1 kb of ram.
- If you wish Mecrisp-Stellaris Forth ran on your favourite chip, just get in contact with Matthias, and maybe there will be another port soon.

What is Forth ?

If you are familiar with Forth already, then bypass this note and go straight to “Quick Start Here” below.

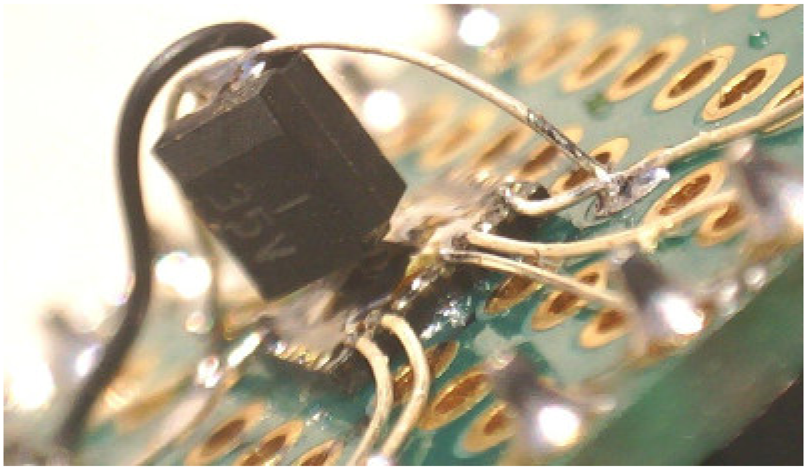
A long Forth Intro

Note

- Forth was invented by Charles Moore in 1968 on an IBM 1130.
- Forth is a program that once flashed into your MCU allows you to **immediately** write and test programs, read Registers, toggle GPIO pins, read the A-D and much, much more. No compiler is needed as it is built into the program, which is flashed onto the MCU. All

you need is a serial terminal to talk to the MCU in Forth.
- Mecrisp-Stellaris Forth can run on a single $0.60 STM32F051 MCU with **zero** external components, but you may need a filter capacitor depending on your wiring layout.

This is a STM32F051-QFN MCU I hand soldered onto 0.1” pitch fibreglass matrix board and flashed with Mecrisp-Stellaris. It has only one filter cap and works perfectly. Those large shiny wires are 0.5m wirewrap wire, exactly the width of the QFN pads.



Quick Start Here

This describes the basic steps to quickly install flash Mecrisp-Stellaris using your choice of methods.

Where to go now ?

Have a look in the Dictionary for all the commands you can use ?

How about running some ‘blinky’ programs on a F0 Discovery Board ?

STM32F051 Discovery Board, about $10

STM32F0-Discovery
STM32 Releasing your creativity
STM32 F0
STMicroelectronics
Everything you need to discover STM32 F0 32-bit ARM Cortex™-M0 based MCUs featuring:
Evaluation board
Embedded ST-LINK/V2
USB interface for debugging and programming
Numerous examples available on www.st.com
www.st.com/stm32f0discovery

Where to go next ?

It’s traditional to build a LED ‘blinky’ as your first embedded program, and we have lots here, check out the Blinky Bazar

How about some working projects ?

You may be wondering, ‘if a chip has a thousand register and register bitfields, how will I *ever* write all those Memory Mapped words?’ Fortunately you don’t have to if you use svd2forth. So why not check it out ?

Into *Object Oriented Programming* ? we have some exiting new developments here

Any other tips ?

Sure, have a browse around this site (if you haven’t already):-

- Read the Forth Intro
- Maybe read some suggestions for Forth problem solving
- Look thru the online book suggestions ?
- There is now a **free Ebook** (epub) of this site, download it **here**

I’ve found some bugs, horrible spelling errors, or have some good tips, corrections etc, who do I contact ?

See the contact list

Common User Issues

Lockups

Problem: You’ve just flashed Mecrisp-Stellaris Forth onto a board or chip and your terminal shows the boot up Greetings message.

When you hit the “Enter” key a “OK” response is received and all looks good. However when you upload Forth source code the terminal locks up after the first few Words requiring the MCU be reset to recover … what’s wrong ??

Answer: Forth has a compiler inside the software, which is inside your MCU and when you enter or upload any source ‘Definitions’ (Words) they are compiled, one after the other.

Now the MCU needs time to compile the Definitions and if new Definitions are uploaded while the chip is still compiling the previous Definition it will appear to ‘lock up’. This is a “Flow Control” issue.

Note

Mecrisp-Stellaris won’t generally ‘lock up’ without giving a error message, of course if you were to write code that disabled the USART, or turned of the CLOCK etc, it may appear that Mecrisp has locked up but this is entirely different to the “Flow Control” problem described above.

Solution 1

The simple way to get around this problem is to use a terminal with a ‘end of line delay’ usually about 200 milliseconds, to allow the MCU time to compile the longest Definitions.

Find how to configure a 200 millisecond “end of line” delay for the terminal you are using and if it doesn’t have one, find one that does.

The downside is that the serial upload speed will be slowed by 200 milliseconds for *EVERY LINE*, long or short.

If you use Linux/*bsd, or Apple Mac see this page serial terminals

Solution 2

A much better but more complex solution is called “Hardware Flow Control”, This solution does not need slow “end of line delays” and compiles as fast as the MCU is able. See hardware handshaking.

GPIO Manipulation

For a beginner trying to enable then manipulate the GPIO Ports on the STM32F103 (Blue Pill) with Mecrisp Stellaris is a monumental task . I’ve just spent all day on this thing trying to get the portB pins flashing a couple of LEDs. The STM32F103 data sheet was written by Boffins for Boffins, the average electronics hobbyist, even some of the better ones would find this document really hard going. For a start it’s over 100 pages (the SMT32F family one is over 1,000 pages) so you are constantly scanning for what you need next.

Solution

See the Blue Pill Starter Kit page.

How To Turnkey

Q: I want to make my program run from cold boot or after a reset how do I do that ?

A: This called “Turnkey” because the user turns a key and it always runs.

If a word named INIT is found in the Dictionary at boot, the INIT Word contents will be executed after reboot ot power up.

Warning

If your program is a **closed loop program with no method to break out of the loop such as “KEY?” then reflashing Mecrisp-Stellaris will be required to access the Terminal because you won’t have access to “ERASEFLASH”**. Make sure you erase ALL the Flash before reflashing Mecrisp-Stellaris as well.

Example 1

Turnkey operation, no way to break out of the “closed.loop.program”

```
compiletoflash

: closed.loop.program      \ runs forever
  begin
  ." This MCU is running a TURNKEY application" cr
  again
;
```

```
: INIT
  closed.loop.program
;

compiletoram
```

Rebooting the MCU will print the above message “This MCU is running a TURNKEY application” again and again … forever.

Example 2

Turnkey with Terminal escape.

```
compiletoflash

: non.closed.loop.program  \ loop until keyboard key pressed
which will exit loop and return control to the terminal
  begin
  ." This MCU is running a TURNKEY application" cr
  key? until
;

: INIT
  non.closed.loop.program
;

compiletoram
```

Rebooting the MCU will print the above message “This MCU is running a TURNKEY application” again and again until a keyboard key is hit.

Different Forths Have Different Words!

It seems there have been many variations of Forth over the years and Words get dropped or changed or modified ?

Actually there is no “Standard Forth” and the designer of Forth, Charles Moore was against any ‘standards’ because he felt they detracted from the ability of Forth to be a ‘problem solving language’. (I hope I have phrased that correctly). Sure there are “Standards Committees” for Forth, but he was not in favor of them.

C.Moore also said when asked “what is Forth?” , “I can’t say for sure what Forth is, but I know it when I see it”.

The Forth community has a saying that sums this up fairly well I think … “When you have learnt ONE Forth, then you have learnt ONE Forth”.

Then when you factor in different MCU’s there are always different designs, features, memory etc, so even within Mecrisp-Stellaris there are some features available on some MCU types, and not on others because of these differences.

The utter malleability of Forth is unique, you won’t find it anywhere else, try this in the safety of your own home!

```
: 1 0 ;
```

or this one

```
: + - ;
```

Note

One thing to be aware off is that people don’t program in Forth, they develop a “Domain Specific Language” to solve their problem with using Forth.

My Solution Isn’t Here!!

1. click on the “index” page link on this site, try a keyword in the “search” box.
2. Have a look thru the Mecrisp “Discussions Site” here https://sourceforge.net/p/mecrisp/discussion/ and if you can’t find anything hopeful post a message there describing your problem. This page is made from such posts and answers. Both the creator of Mecrisp and myself check these pages every few days.
3. Contact Us

> [What’s New ?](#) <

Dictionary

Example Dictionary Listing

```
Address: 0000DFD4 Link: 0000DFEC Flags: 00000040 Code: 0000DFE4 Name: pull-down
Address: 0000DFEC Link: 0000E024 Flags: 00001820 Code: 0000DFFC Name: green-on
Address: 0000E024 Link: 0000E05C Flags: 00001820 Code: 0000E034 Name: green-off

see green-on

0000DFFC: 2090  movs r0 #90
0000DFFE: 04C0  lsls r0 r0 #13
0000E000: 3080  adds r0 #80
0000E002: 0100  lsls r0 r0 #4
0000E004: 6983  ldr r3 [ r0 #18 ]
0000E006: 2280  movs r2 #80
0000E008: 0092  lsls r2 r2 #2
0000E00A: 4313  orrs r3 r2
0000E00C: 6183  str r3 [ r0 #18 ]
0000E00E: 4770  bx lr
```

Forth Definition Components

```
Address  Link  Flags  Code  Name
```

Word Header Fields

Name	Description
ADDRESS	Start of the definition block, always 16 Bytes before the CODE section
LINK	Address of the next definition, the one after this one
FLAGS	see “FLAG Codes” below
CODE	The address of the start of the executable code for this Word. Call this address to execute this Word.
NAME	The name of the definition (Word)

FLAG Codes

Taken from: mecrisp-stellaris-source/common/datastackandmacros.s

Flag	RA Only?	Description
0x0000		Visible, non-immediate
0x0008		Opcodable; 0: special case 1: plusminus 2: computational logic 3: DC unequal 4: push 5: memory Write,
0x0010		Immediate
0x0020		Inline - that means, it’s contents will be inlined until $4130, the ret opcode is found. On ARM, this is either $4770 bx lr or $BD00 pop { pc } .
0x0030		Immediate+ Inline means: Immediate, Compile-Only
0x0080		Ramallot means that RAM is reserved and initialised by catchflashpointers for this definition on startup
0x0040		0-foldable, e.g. constants
0x0041		1-foldable, e.g. 1+
0x0042		2-foldable, e.g. swap, xor
0x0043		3-foldable
0x0044		4-foldable
0x0045		5-foldable

Flag	RA Only?	Description
0x0046		6-foldable
0x0047		7-foldable
0x0100		buffers in compiletoflash mode
0x0200	Y	Allocator, this definition has individal optimisation code and a second entry point for the optimiser after the first bx lr or pop { pc } opcode.
0x0400	Y	Jump Eaters, this definition can handle cached jump instructions itself, for merging of comparisons with control structures.
0x0800	Y	Bxlr, is used once for ; and means that despite ; is immediate we can keep the direct bx lr return without the need to write the push { lr } opcode first for a later pop { pc } . Usually, the optimiser falls back to the push { lr } … pop { pc } sequence as soon as a call is compiled or something flagged as immediate is encountered.
0x1000	Y	Inlinecache, denotes that this definition carries an automatically generated inline cache table, which allows recompilation of this definition elsewhere with local optimisations in place.
0x2000	Y	Literator, never shows up in the dictionary, but appears in inline cache table to denote literals instead of code entry point with flag pairs.
0x4000	Y	User, is unused in all Mecrisp-Stellaris cores. This flag is free for the user, and the loadable vocabulary extension relies on it.

Flag	RA Only?	Description
0xFFFF		Invisible

RA Core Register Usage

Register Purpose Notes

r0 Free scratch register Saved on interrupt entry by hardware r1 Free scratch register Saved on interrupt entry by hardware r2: Free scratch register Saved on interrupt entry by hardware r3: Free scratch register Saved on interrupt entry by hardware

r4: Inner loop count Needs Push and Pop when used otherwise r5: Inner loop limit Needs Push and Pop when used otherwise

r6=TOS: Top-Of-Stack Stack design is interrupt safe) r7=PSP: Parameter Stack Pointer Stack design is interrupt safe)

r8: Unused r9: Unused r10: Unused r11: Unused r12: Unused Saved on interrupt entry by hardware

r13=SP: Return Stack Pointer r14=LR: Link Register r15=PC: Program Counter, always odd

Number Bases

prefix	Base	***exactly ANS***	
%	binary	(- -)	Sets base to 2
#	decimal	(- -)	Sets base to 10
$	hex	(- -)	Sets base to 16
	base	(- - a-addr)	Base variable address

Data Types

Name	Bits	Description
Word	32	Native Mecrisp-Stellaris data length
Cell	32	Same as Word above
Half Word	16	Half length
Character	8	One Byte
Bit	1	Smallest

Key to Tables

Phrase	Description	Comment	Examples
n, n1,	single-length signed integer	-2147483648 to 2147483647	$80000000 . -2147483648, $7FFFFFFF . 2147483647
d, d1,	double-length signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	$FFFFFFFF $FFFFFFFF m* d. 1 ok. (-1 * -1 = 1)
u, u1,	single-length unsigned integer	0 to 4294967295	$FFFFFFFF u. 4294967295,
ud, ud1	unsigned double-length integer	0 to 18,446,744,065,119,617,025	$FFFFFFFF $FFFFFFFF um* ud. 18446744065119617025

Phrase	Description	Comment	Examples
c	character (byte)	8 bits, 0 to 255	$FF
h	halfword	16 bits, 0 to 65,535	$FFFF
x	cell		logic, memory operations, logging, bitmaps …
addr	address		
c-addr	character address		
R:	return stack		
flag	Binary 1 or 0		
“name”	input stream token		

Compiler Error Messages

Derived by grepping the source for “write”, “writeln” and “Fehler” (error).

Message	Source	Comments
Jump too far		Dreaded conditional jump range limitation. Workaround: create new Word with offending code.
Cannot write into core !	flash16bytesblockwrite.s	

Message	**Source**	**Comments**
Cannot write into core !	flash.s	
Create needs name !	compiler-flash.s	Often caused by a lone “;”
Flash full	compiler-flash.s	The chip Flash Memory is full.
is compile-only	interpreter.s	Cannot be used interactively, must be inside a Definition (Word)
Jump too far	controlstructures.s	All jumps use Thumb 1 short opcodes. Solution; split your code into smaller Words
Not enough RAM	compiler-flash.s	Not enough RAM is available to complete the Word
not found.	interpreter.s	Name of word, constant, variable, etc cannot be found, usually a source typo
ok.	interpreter.s	All is good!
Ram full	compiler-flash.s	The chip RAM is full
Redefine	compiler-flash.s	A Word, constant, variable, etc has been defined more than once
Stack underflow	interpreter.s	The Data Stack is empty, you tried to pop data off it, but there is no data there

Message	**Source**	**Comments**
Stack not balanced.	compiler.s	The Return Stack is unbalanced, use of >R and R> was not equal
Stack overflow	interpreter.s	The Data Stack is full, default size is 64 elements, same as the Return Stack.
Structures don’t match	controlstructures.s	Example: a missing “then” from a “if else then” statement
Unhandled Interrupt !	interrupts-common.s	A Interrupt has been triggered, but no code is handling it. See Exceptions
Variables collide with dictionary	compiler-flash.s	
Wrong address or data for writing flash !	flash.s for your MCU	See notes

Terminal-IO

Exactly ANS, some logical extension

emit?	(- - Flag)	Ready to send a character ?
key?	(- - Flag)	Checks if a key is waiting
key	(- - Char)	Waits for and fetches the pressed key
emit	(Char - -)	Emits a character

Exactly ANS, some logical extension

hook-emit? hook-key? hook-key hook-emit	(- - a-addr)	Hooks for redirecting terminal IO on the fly
serial-emit? serial-key?	(- - Flag)	Serial interface terminal routines as default communications
serial-key	(- - Char)	Waits for and fetches the pressed key
serial-emit	(Char - -)	Emits a character
hook-pause	(- - a-addr)	Hook for a multitasker
pause	(- -)	Task switch, none for default

Character Manipulation

How to convert numbers to characters and convert characters to numbers.

Command	Stack Notation	Description	Code or Comment	Result
[char] *	(- - char)	Compiles code of following char when executed	When used in a definition (function, Word)	

Command	**Stack Notation**	**Description**	**Code or Comment**	**Result**
char *	(- - char)	Gives code of following char	char 3 hex.	00000033 ok.
emit	(char - -)	Emits a character	$33 emit	3 ok.

Stack Jugglers

Single-Jugglers

Exactly ANS, some logical extension			**RA Core Only ?**
depth	(- - +n)	Gives number of single-cell stack items.	
nip	(x1 x2 - - x2)	Drop the first item below the top of the stack	
drop	(x - -)		
rot	(x1 x2 x3 - - x2 x3 x1)		
-rot	(x1 x2 x3 - - x3 x1 x2)		
swap	(x1 x2 - - x2 x1)		
tuck	(x1 x2 - - x2 x1 x2)		
over	(x1 x2 - - x1 x2 x1)		

Exactly ANS, some logical extension			**RA Core Only ?**
?dup	(x - - 0 | x x)		
dup	(x - - x x)		
pick	(… xi+1 xi … x1 x0 i - - … x1 x0 xi)	Copy a element X levels down the stack to the top of the stack	
>r	(x - -) (R: - - x)	Push to Return Stack (the Return Stack must be balanced)	
r>	(- - x) (R: x - -)	Pull from Return Stack	
r@	(- - x) (R: x - - x)	Copy from Return Stack	
rdrop	(- -) (R: x - -)		
rdepth	(- - +n)	Gives number of return stack items.	
rpick	(i - - xi) R: (… xi … x0 - - … xi … x0)		
roll	(xu xu-1 … x0 u – xu-1 … x0 xu)	Remove u. Rotate u+1 items on the top of the stack.	Yes
-roll	(xu-1 … x0 xu u – xu xu-1 … x0 u)		Yes

Double-Jugglers

They perform the same for double numbers

2nip	(x1 x2 x3 x4 - -x3 x4)
2drop	(x1 x2 - -)
2rot	(x1 x2 x3 x4 x5 x6 - - x3 x4 x5 x6 x1 x2)
2-rot	(x1 x2 x3 x4 x5 x6 - - x5 x6 x1 x2 x3 x4)
2swap	(x1 x2 x3 x4 - - x3 x4 x1 x2)
2tuck	(x1 x2 x3 x4 - - x3 x4 x1 x2 x3 x4)
2over	(x1 x2 x3 x4 - - x1 x2 x3 x4 x1 x2)
2dup	(x1 x2 - - x1 x2 x1 x2)
2>r	(x1 x2 - -) (R: - -x1 x2)
2r>	(- - x1 x2) (R: x1 x2 - -)
2r@	(- - x1 x2) (R: x1 x2 - -x1 x2)
2rdrop	(- -) (R: x1 x2 - -)

Stack pointers

Stack Pointers

sp@	(- - a-addr)	Fetch data stack pointer

Stack Pointers

sp!	(a-addr - -)	Store data stack pointer
rp@	(- - a-addr)	Fetch return stack pointer
rp!	(a-addr - -)	Store return stack pointer

Logic

exactly ANS, some logical extensions

arshift	(x1 u - - x2)	Arithmetic right-shift of u bit-places
rshift	(x1 u - - x2)	Logical right-shift of u bit-places
lshift	(x1 u - - x2)	Logical left-shift of u bit-places
shr	(x1 - - x2)	Logical right-shift of one bit-place
shl	(x1 - - x2)	Logical left-shift of one bit-place
ror	(x1 - - x2)	Logical right-rotation of one bit-place
rol	(x1 - - x2)	Logical left-rotation of one bit-place
bic	(x1 x2 - - x3)	Bit clear, identical to “not and”
not	(x1 - - x2)	Invert all bits

exactly ANS, some logical extensions		
xor	(x1 x2 - - x3)	Bitwise Exclusive-OR
or	(x1 x2 - - x3)	Bitwise OR
and	(x1 x2 - - x3)	Bitwise AND
false	(- - 0)	False-Flag
true	(- - -1)	True-Flag
clz	(x1 - - u)	Count leading zeros

Single Number Math

exactly ANS	***some logical extensions***	
u/mod	(u1 u2 - - u3 u4)	32/32 = 32 rem 32 Division u1 / u2 = u4 remainder u3
/mod	(n1 n2 - - n3 n4)	n1 / n2 = n4 rem n3
mod	(n1 n2 - - n3)	n1 / n2 = remainder n3
/	(n1 n2 - - n3)	n1 / n2 = n3
*	(u1 n1 u2 n2 - - u3 n3)	32*32 = 32 Multiplication
min	(n1 n2 - - n1 n2)	Keeps smaller of top two items

exactly ANS	***some logical extensions***	
max	(n1 n2 - - n1 n2)	Keeps greater of top two items
umin	(u1 u2 - - u1 u2)	Keeps unsigned smaller
umax	(u1 u2 - - u1 u2)	Keeps unsigned greater
2-	(u1 n1 - - u2 n2)	Subtracts two, optimized
1-	(u1 n1 - - u2 n2)	Subtracts one, optimized
2+	(u1 n1 - - u2 n2)	Adds two, optimized
1+	(u1 n1 - - u2 n2)	Adds one, optimized
even	(u1 n1 - - u2 n2)	Makes even. Adds one if uneven.
2*	(n1 - - n2)	Arithmetric left-shift
2/	(n1 - - n2)	Arithmetric right-shift
abs	(n - - u)	Absolute value
negate	(n1 - - n2)	Negate
-	(u1|n1 u2|n2 - - u3|n3)	Subtraction
+	(u1|n1 u2|n2 - - u3|n3)	Addition

Double Number Math

exactly ANS, some logical extensions

um*	(u1 u2 - - ud)	32*32 = 64 Multiplication
ud*	(ud1 ud2 - - ud3)	64*64 = 64 Multiplication
udm*	(ud1 ud2 - - ud3-Low ud4-High)	64*64=128 Multiplication
um/mod	(ud u1 - - u2 u3)	ud / u1 = u3 remainder u2
ud/mod	(ud1 ud2 - - ud3 ud4)	64/64 = 64 rem 64 Division ud1 / ud2 = ud4 remainder ud3
m*	(n1 n2 - - d)	n1 * n2 = d
m/mod	(d n1 - - n2 n3)	d / n1 = n3 remainder r2
d/mod	(d1 d2 - - d3 d4)	d1 / d2 = d4 remainder d3
d/	(d1 d2 - - d3)	d1 / d2 = d3
*/	(n1 n2 n3 - - n4)	n1 * n2 / n3 = n4
u*/	(u1 u2 u3 - - u4)	u1 * u2 / u3 = u4
*/mod	(n1 n2 n3 - - n4 n5)	n1 * n2 / n3 = n5 remainder n4
u*/mod	(u1 u2 u3 - - u4 u5)	u1 * u2 / u3 = u5 remainder u4
d2*	(d1 - - d2)	Arithmetric left-shift
d2/	(d1 - - d2)	Arithmetric right-shift

exactly ANS, some logical extensions

dshl	(ud1 - - ud2)	Logical left-shift, same as d2*
dshr	(ud1 - - ud2)	Logical right-shift
dabs	(d - - ud)	Absolute value
dnegate	(d1 - - d2)	Negate
d-	(ud1|d1 ud2|d2 - - ud3|d3)	Subtraction
d+	(ud1|d1 ud2|d2 - - ud3|d3)	Addition
s>d	(n - - d)	Makes a signed single number double length

Fixed Point Arithmetic

See also

Fixed Point Page Fixed Point Examples

s31.32 fixed point numbers, speciality!

Fixpoint numbers are stored (n-comma n-whole) and can be handled like signed double numbers

d-	(df1 df2 - - df3)	Addition of two fixpoint numbers
d+	(df1 df2 - - df3)	Subtraction of two fixpoint numbers

Fixpoint numbers are stored (n-comma n-whole) and can be handled like signed double numbers

f/	(df1 df2 - - df3)	Division of two fixpoint numbers
f*	(df1 df2 - - df3)	Multiplication
hold<	(char - -)	Adds character to pictured number output buffer from behind
f#S	(n-comma1 - - n-comma2)	Adds 32 comma-digits to number output
f#	(n-comma1 - - n-comma2)	Adds one comma-digit to number output
f.	(df - -)	Prints a fixpoint number with 32 fractional digits
f.n	(df n - -)	Prints a fixpoint number with n fractional digits
number	(c-addr length - - 0)	Tries to convert a string to a number. 0 = FAILED
	(c-addr length - - n 1)	1 = success, a 32 byte number is on the STACK
	(c-addr length - - n-low n-high 2)	2 = success, TWO 32 bit numbers, a HIGH BYTE and a LOW BYTE on the STACK

Decision Making

Single-Comparisons

exactly ANS, some logical extensions

Unigned Comparisons		
u<=	(u1 u2 - - flag)	u1 less-or-equal u2
u>=	(u1 u2 - - flag)	u1 greater-or-equal u2
u>	(u1 u2 - - flag)	u1 greater-than u2
u<	(u1 u2 - - flag)	u1 less-than u2

Signed Comparisons

exactly ANS, some logical extensions

<=	(n1 n2 - - flag)	n1 less-or-equal n2
>=	(n1 n2 - - flag)	n1 greater-or-equal n2
>	(n1 n2 - - flag)	n1 greater-than n2
<	(n1 n2 - - flag)	n1 less-than n2
0<	(n - flag)	n less than zero
0<>	(n - - flag)	n not equal to zero
0=	(n - - flag)	n equals zero
<>	(n1 n2 - - flag)	n1 not-equals n2

exactly ANS, some logical extensions

=	(n1 n2 - - flag)	n1 equals n2

Double-Comparisons

They perform the same for double numbers.

du>	(ud1 ud2 - - flag)
du<	(ud1 ud2 - - flag)
d>	(d1 d2 - - flag)
d<	(d1 d2 - - flag)
d0<	(d - - flag)
d0=	(d - - flag)
d<>	(d1 d2 - - flag)
d=	(d1 d2 - - flag)

Bits: Test, Toggle, Clear and Set

Note: TRUE = 1, FALSE = 0

Word	Stack Comment	Description
cbit@	(mask c-addr - - flag)	Test BIts in byte-location

Word	**Stack Comment**	**Description**
hbit@	(mask a-addr -- flag)	Test BIts in halfword-location
bit@	(mask a-addr -- flag)	Test Bits in word-location, If a Bit is HIGH it leaves TRUE on the stack, otherwise LOW leaves FALSE
cxor!	(mask c-addr --)	Toggle bits in byte-location
hxor!	(mask a-addr --)	Toggle bits in halfword-location
xor!	(mask a-addr --)	Toggle bits in word-location
cbic!	(mask c-addr --)	Clear BIts in byte-location
hbic!	(mask a-addr --)	Clear BIts in halfword-location
bic!	(mask a-addr --)	Clear BIts in word-location
cbis!	(mask c-addr --)	Set BIts in byte-location
hbis!	(mask a-addr --)	Set BIts in halfword-location

Word	Stack Comment	Description
bis!	(mask a-addr - -)	Set BIts in word-location

Memory Status

Word	Stack Comment	Description
unused	(- - free memory)	Displays memory depending on compile mode (Ram or Flash)

Memory Access

- subtle differences to ANS, special cpu-specific extensions
- Fetch-Modify-Store operations like + ! or bic! are NOT ATOMIC.

move	(c-addr1 c-addr2 u - -)	Moves u Bytes in Memory
fill	(c-addr u c)	Fill u Bytes of Memory with value c
2constant name	(ud|d - -)	Makes a double constant.
constant name	(u|n - -)	Makes a single constant. i.e. “$1024 constant one-kb”
2variable name	(ud|d - -)	Makes an initialized double variable
variable name	(n|n - -)	Makes an initialized single variable. i.e. “0 variable one-kb”

nvariable name	(n1*u|n n1 - -)	Makes an initialized variable with specified size of n1 words Maximum is 15 words
buffer: name	(u - -)	Creates a buffer in RAM u bytes in length
2@	(a-addr - - ud|d)	Fetches double number from memory
2!	(ud|d a-addr - -)	Stores double number in memory
@	(a-addr - - u|n)	Fetches single number from memory
!	(u|n a-addr - -)	Stores single number in memory
+!	(u|n a-addr - -)	Plus Store, use to increment a variable, register etc
h@	(c-addr - - char)	Fetches halfword from memory
h!	(char c-addr)	Stores halfword in memory
h+!	(u|n a-addr - -)	Add to halfword memory location
c@	(c-addr - - char)	Fetches byte from memory
c!	(char c-addr)	Stores byte in memory
c+!	(u|n a-addr - -)	Add to byte memory location

String Formatting

String Routines

Warning

” may not render properly on your browser, hence copying and pasting commands that use ” may fail. Click on “Show Source” to see the correct character.

exactly ANS, some logical extensions

type	(c-addr length - -)	Prints a string.
s” Hello”	(- - c-addr length)	Compiles a string and gives back its address and length when executed
.” Hello”	(- -)	Compiles a string and prints it when executed.
(Comment)	Ignore Comment	
\ Comment	Comment to end of line	
cr	(- -) Emits line feed	
bl	(- - 32) ASCII for Space	
space	(- -) Emits space	
spaces	(n - -)	Emits n spaces if n is positive
compare	(caddr-1 len-1 c-addr-2 len-2 - - flag)	Compares two strings

exactly ANS, some logical extensions		
accept	(c-addr maxlength - - length)	Read input into a string.

Counted String Routines

Warning

” may not render properly on your browser, hence copying and pasting commands that use ” may fail. Click on “Show Source” to see the correct character.

exactly ANS, some logical extensions		
ctype	(cstr-addr - -)	Prints a counted string.
c” Hello”	(- - cstr-addr)	Compiles a counted string and gives back its address when executed.
cexpect	(cstr-addr maxlength - -)	Read input into a counted string.
count	(cstr-addr - - c-addr length)	Convert counted string into addr-length string
skipstring	(cstr-addr - - a-addr)	Increases the pointer to the aligned end of the string.

Pictured Numerical Output

Pictured Numerical output		
.digit	(u - - char)	Converts a digit to a char

Pictured Numerical output

digit	(char - - u true | false)	Converts a char to a digit
[char] *	(- - char)	Compiles code of following char when executed
char *	(- - char)	gives code of following char
hold	(char - -)	Adds character to pictured number output buffer from the front.
hold<	(char - -)	Adds character to pictured number output buffer from behind
sign	(n - -)	Add a minus sign to pictured number output buffer, if n is negative
#S	(ud1|d1 - - 0 0)	Add all remaining digits from the double length number to output buffer
f#S	(n-comma1 - - n-comma2)	Adds 32 comma-digits to number output
#	(ud1|d1 - - ud2|d2)	Add one digit from the double length number to output buffer
f#	(n-comma1 - - n-comma2)	Adds one comma-digit to number output
#>	(ud|d - - c-addr len)	Drops double-length number and finishes pictured numeric output ready for type
<#	(- -)	Prepare pictured number output buffer

Pictured Numerical output

u.	(u - -)	Print unsigned single number
.	(n - -)	Print single number
ud.	(ud - -)	Print unsigned double number
d.	(d - -)	Print double number

Deep Insights

Deep Insights

words	(- -)	Prints list of defined words and properties
list	(- -)	Prints all defined words. From; dissasembler-mx.txt
.s	(many - - many)	Prints stack contents, signed
u.s	(many - - many)	Prints stack contents, unsigned
h.s	(many - - many)	Prints stack contents, unsigned, hex
hex.	(u - -)	Prints 32 bit unsigned in hex base, needs emit only. This is independent of number subsystem.

User Input and Interpretation

exactly ANS, some logical extensions

exactly ANS, some logical extensions

query	(- -)	Fetches user input to input buffer
tib	(- - cstr-addr)	Input buffer
current-source	(- - addr)	Double-Variable which contains source
tsource	(c-addr len - -)	Change source
source	(- - c-addr len)	Current source
>in	(- - addr)	Variable with current offset into source
token	(- - c-addr len)	Cuts one token out of input buffer
parse	(char - - c-addr len)	Cuts anything delimited by char out of input buffer
evaluate	(any addr len - - any)	Interpret given string
interpret	(any - - any)	Execute, compile, fold, optimize…
quit	(many - -) (R: many - -)	Resets Stacks
hook-quit	(- - a-addr)	Hook for changing the inner quit loop

Dictionary Expansion

exactly ANS, some logical extension

exactly ANS, some logical extension

align	(- -)	Aligns dictionary pointer
aligned	(c-addr - - a-addr)	Advances to next aligned address
cell+	(x - - x+4)	Add size of one cell
cells	(n - - 4*n)	Calculate size of n cells
allot	(n - -)	tries to advance Dictionary Pointer by n bytes, aborts, if not enough space available
here	(- - a-addr|c-addr)	Gives current position in Dictionary
,	(u|n - -)	appends a single number to dictionary
><,	(u|n - -)	reverses high and low-halfword, then appends it to dictionary
h,	(u|n - -)	appends a halfword to dictionary
compiletoram?	(- - ?)	currently compiling into ram ?
compiletoram	(- -)	makes ram the target for compiling
compiletoflash	(- -)	makes flash memory the target for compiling
forgetram	(- -)	Forget definitions in ram without a reset

Speciality!

Words	Stack Comment	Description
string,	(c-addr len - -)	Inserts a string of maximum 255 characters without runtime
literal,	(u|n - -)	Compiles a literal with runtime
inline,	(a-addr - -)	Inlines the choosen subroutine
call,	(a-addr - -)	Compiles a call to a subroutine
jump,	(Hole-for-Opcode Destination)	Writes an unconditional Jump to a-addr-Destination with the given Bitmask as Opcode into the halfword sized a-addr-Hole
cjump,	(Hole-for-Opcode Destination Bitmask)	Writes a conditional Jump to a-addr-Destination with the given Bitmask as Opcode into the halfword sized a-addr-Hole
ret,	(- -)	compiles a ret opcode
flashvar-here	(- - a-addr)	Gives current RAM management pointer
dictionarystart	(- - a-addr)	Current entry point for dictionary search
dictionarynext	(a-addr - - a-addr flag)	Scans dictionary chain and returns true if end is reached.

Special Words Depending on MCU Capabilities

Words	**Stack Comment**	**Description**
c,	(char - -)	Appends a byte to dictionary. NOT AVAILABLE ON ALL MCU’S. Check with ‘ c, or just look in ‘words’.
halign	(- -)	Makes Dictionary Pointer even, if uneven.
movwmovt,	(x Register - -)	Generate a movw/movt-Sequence to get x into any given Register. M3/M4 only
registerliteral,	(x Register - -)	Generate shortest possible sequenceto get x into given low Register. On M0: A movs-lsls-adds… sequence M3/M4: movs / movs-mvns / movw / movw-movt
12bitencoding	(x - - x false | bitmask true)	Can x be encoded as 12-bit immediate ?
eraseflash	(- -)	Erases everything after — Flash Dictionary — Clears Ram, Restarts Forth. Does not erase Mecrisp-Stellaris
eraseflashfrom	(a-addr –)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr –)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr –)	Writes halfword to flash

Flags and Inventory

Speciality!	Stack Comment	Description
smudge	(- -)	Makes current definition visible, burns collected flags to flash and takes care of proper ending
inline	(- -)	Takes the code of a word, and puts it in place of a call to this word
immediate	(- -)	Will not compile this word into the dictionary but execute the word immediately
compileonly	(- -)	Makes current definition compileonly
setflags	(char - -)	Sets Flags with a mask. This isn't immediate,
(create) name	(- -)	Names a location; space may be allocated at this location, or it can be set to contain a string or other initialized value. Instance behavior returns the address of the beginning of this space. Use FIG-style <builds .. does> !
find	(c-addr len - - a-addr flags)	Searches for a String in Dictionary. Gives back flags, which are different to ANS!

Folding

Speciality!	Stack Comment	Description	Applies To
0-foldable	(- -)	Current word becomes foldable with 0 constants	constants variables [‘] [char]
1-foldable	(- -)	1 constant	?dup drop negate

Speciality!	**Stack Comment**	**Description**	**Applies To**
2-foldable	(- -)	2 constants	+ - * swap nip
3-foldable	(- -)	3 constants	rot
4-foldable	(- -)	4 constants	d+ d- 2swap
5-foldable	(- -)	5 constants	
6-foldable	(- -)	6 constants	
7-foldable	(- -)	7 constants	

Compiler Essentials

The true POWER of Forth resides in the Words listed in this table.

subtle differences to ANS		
execute	(a-addr - -)	Calls subroutine
recurse	(- -)	Lets the current definition call itself
‘ name	(- - a-addr)	Tries to find name in dictionary gives back executable address
[‘] name	(- - a-addr)	Tick that compiles the executable address of found word as literal
postpone	(- -)	See Glossary

subtle differences to ANS

does>	(- -)	executes: (- -a-addr) Gives address to where you have stored data.
<builds	(- -)	Makes Dictionary header and reserves space for special call.
create name	(- -)	Create a definition with default action which cannot be changed later. Use <builds does> instead. Equivalent to : create <builds does> ;
state	(- - a-addr)	Address of state variable
]	(- -)	Switch to compile state
[	(- -)	Switch to execute state
;	(- -)	Finishes new definition
: name	(- -)	Opens new definition

Control Structures

Internally, they have complicated compile-time stack effects.

Summary

Structure	Description
DO ... LOOP	Finite loop incrementing by 1

Structure	Description
DO … +LOOP	Finite loop incrementing by X
BEGIN … UNTIL	Indefinite loop terminating when is ‘true’
BEGIN … WHILE … REPEAT	Indefinite loop terminating when is ‘false’
BEGIN … AGAIN	Infinite loop
IF … ELSE … THEN	Two-branch conditional; performs words following IF it is ‘true’ and words following ELSE if it is ‘false’. THEN marks the point at which the paths merge.
IF … THEN	Like the two-branch conditional, but with only a ‘true’ clause.

Decisions

Decisions		***exactly ANS***
then	(- -)	This is the common flag if … [else …] then structure.
else	(- -)	flag if … [else …] then
if	(flag - -)	structure.

Syntax

“IF that’s the case, do this, ELSE do that … and THEN continue with …”

```
flag if ... then
flag if ... else ... then
```

Case

Case		***exactly ANS***
case	(n - -n)	Begins case structure
of	(m - -)	Compares m with n, choose this if n=m
?of	(flag - -)	Flag-of, for custom comparisions
endof	(- -)	End of one possibility
endcase	(n - -)	Ends case structure, discards n

Syntax

```
n case
     m1    of  ... endof
     m2    of  ... endof
     flag  ?of ... endof
     all others
  endcase
```

Indefinite Loops

A loop structure in which the words contained within the loop continue to repeat until some truth condition changes state (true-to-false or false-to-true). In Forth, the indefinite loops begin with the word BEGIN.

Indefinite Loops	**Stack Comment**	**Exactly ANS**
repeat	(- -)	Finish of a middle-flag-checking loop.
while	(flag - -)	Check a flag in the middle of a loop
until	(flag - -)	Begin ... flag until loops as long flag is true

Indefinite Loops	Stack Comment	Exactly ANS
again	(- -)	begin ... again is an endless loop
begin	(- -)	

Syntax

```
begin ... again
begin ... flag until
begin ... flag while ... repeat
```

Definite Loops

A loop structure in which the words contained within the loop repeat a definite number of times. In Forth, this number depends on the starting and ending counts (index and limit) which are placed on the stack prior to the execution of the word DO.

Definite Loops	Stack Comment	Exactly ANS
i	(- -u|n)	Gives innermost loop index
j	(- -u|n)	Gives second loop index
k	(- -u|n)	Gives third loop index
unloop	(R: old-limit old-index - -)	Drops innermost loop structure, pops back old loop structures to loop registers
exit	(- -)	Returns from current definition. Compiles a ret opcode.

Definite Loops	Stack Comment	Exactly ANS
leave	(- -) (R: old-limit old-index - -)	Leaves current innermost loop promptly
+loop	(u|n - -) (R: unchanged | old-limit old-index - -)	If you want the index to go up by some number other than one each time around.
loop	(- -) (R: unchanged | old-limit old-index - -)	Increments current loop index register by one and checks whether to continue or not.
?do	(Limit Index - -) (R: unchanged | - -old-limit old-index)	Begins a loop if limit and index are not equal
do	(Limit Index - -) (R: - -old-limit old-index)	Begins a loop

Syntax

```
limit index do ... loop
```

Common Hardware Access

reset	(- -)	Reset on hardware level
dint	(- -)	Disables Interrupts
eint	(- -)	Enables Interrupts

eint?	(- -)	Are Interrupts enabled ?
nop	(- -)	No Operation. Hook for unused handlers !
ipsr	(- - ipsr)	Interrupt Program Status Register
unhandled	(- -)	Message for unhandled interrupts.
irq-systick	(- - a-addr)	Memory locations for IRQ-Hooks
irq-fault	(- - a-addr)	For all faults
irq-collection	(- - a-addr)	Collection of all unhandled interrupts

About This Site

- Now hosted on Sourceforge so you may see adverts placed by Sourceforge etc.

About the Author

I’m a 65 year old electronics technician with 47 years industry experience. I started learning Forth in April 2014 and soon adopted it for all my embedded hardware needs.

This site grew from my extensive notes which I made public because there really isn’t much information about modern Forth available on-line these days. I try to keep errors to a minimum but you know how it is …

Contact The Author

Note

Ways to contact the author

- IRC: Freenode: author may be found on IRC and uses the nickname of “tp”, “tpbsd”
- IRC: irc.freenode.net
- IRC: Freenode channels #mecrisp
- Email <terry@tjporter.com.au>

Note

Contacts regarding Mecrisp-Stellaris Forth, Assembly Language, STM32F projects, bugs, errors, patches, and document suggestions etc, are *always welcome*.

Documentation Copyright

Copyright Terry Porter 2019 terry@tjporter.com.au released under the terms of the FreeBSD Copyright 1992-2019 The FreeBSD Project.The FreeBSD Copyright

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE/DOCUMENTATION IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the FreeBSD Project.

Program Copyrights

Are listed per program and depend on author etc.

DOCUMENTATION FOR MCU(s)

Note

The amount of documentation for all ARM chips is daunting for newcomers. The documentation for older microcontrollers such as the 6800, Z80 or 6502 easily fit in single large document, but as chips have evolved so has the documentation grown. The total documentation is especially hard to grasp for ARM based chips since it consists of documents from the IC manufacturer and documents from CPU core vendor (ARM Holdings). A typical top-down documentation tree is:

- manufacturer website, manufacturer marketing slides so you know what to buy.
- manufacturer electrical datasheet for the exact physical chip.
- manufacturer detailed reference manual that describes common peripherals and aspects of a physical chip family.
- ARM core generic user guide.
- ARM core technical reference manual.
- ARM architecture reference manual that describes the instruction set(s).

Target literature packages for most MCU’s supported by Mecrisp-Stellaris can be found in the download list here:

https://sourceforge.net/projects/mecrisp/files/

Search for “Target literature package”

Example

Target literature package for STM32L053.tar.gz

https://sourceforge.net/projects/mecrisp/files/Target%20literature%20package%20for%20STM32L053.tar.gz/download

Alternate Functions

The I/O pins on any STM32 have great flexibility, and are configured via the “alternate functions” registers. As every model of STM32F chip has different numbers of pins and internal peripherals, the “alternate functions” matrix is only found in the data sheet, not in the programming references.

To access the alternate functions for *each* GPIO, **alternate function mode** must first be enabled via the MODER registers.

GPIOX_MODER (STM32F051)

Pin Function	Code
Input mode (reset state)	00
General purpose output mode	01
Alternate function mode	**10**
Analog mode	11

After the **alternate function mode** is enabled above, the **alternate functions** are then selected per GPIO in the GPIOx_AFRL and GPIOx_AFRH registers.

GPIOA Alternate Function Matrix (STM32F051)

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF
PA0		USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1			
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2			
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3			
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4			
PA4	SPI1_NSS, I2S1_WS	USART2_CK		TSC_G2_IO1	TIM14_CH1		

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF
PA5	SPI1_SCK, I2S1_CK	CEC	TIM2_CH1_ETR	TSC_G2_IO2			
PA6	SPI1_MISO, I2S1_MCK	TIM3_CH1	TIM1_BKIN	TSC_G2_IO3		TIM16_CH1	EV
PA7	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM1_CH1N	TSC_G2_IO4	TIM14_CH1	TIM17_CH1	EV
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT			
PA9	TIM15_BKIN	USART1_TX	TIM1_CH2	TSC_G4_IO1			
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	TSC_G4_IO2			
PA11	EVENTOUT	USART1_CTS	TIM1_CH4	TSC_G4_IO3			
PA12	EVENTOUT	USART1_RTS	TIM1_ETR	TSC_G4_IO4			
PA13	SWDIO	IR_OUT					
PA14	SWCLK	USART2_TX					
PA15	SPI1_NSS, I2S1_WS	USART2_RX	TIM2_CH1_ETR	EVENTOUT			

GPIOB Alternate Function Matrix (STM32F051)

Pin name	AF0	AF1	AF2	AF3
PB0	EVENTOUT	TIM3_CH3	TIM1_CH2N	TSC_G3_IO2
PB1	TIM14_CH1	TIM3_CH4	TIM1_CH3N	TSC_G3_IO3
PB2				TSC_G3_IO4

Pin name	AF0	AF1	AF2	AF3
PB3	SPI1_SCK, I2S1_CK	EVENTOUT	TIM2_CH2	TSC_G5_IO1
PB4	SPI1_MISO, I2S1_MCK	TIM3_CH1	EVENTOUT	TSC_G5_IO2
PB5	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM16_BKIN	I2C1_SMBA
PB6	USART1_TX	I2C1_SCL	TIM16_CH1N	TSC_G5_IO3
PB7	USART1_RX	I2C1_SDA	TIM17_CH1N	TSC_G5_IO4
PB8	CEC	I2C1_SCL	TIM16_CH1	TSC_SYNC
PB9	EVENTOUT	I2C2_SDA	TIM2_CH4	TSC_G6_IO1
PB10	SPI2_NSS	EVENTOUT	TIM1_BKIN	TSC_G6_IO2
PB11	SPI2_SCK		TIM1_CH1N	TSC_G6_IO3
PB12	SPI2_NSS	EVENTOUT	TIM1_BKIN	TSC_G6_IO2
PB13	SPI2_SCK		TIM1_CH1N	TSC_G6_IO3
PB14	SPI2_MISO	TIM15_CH1	TIM1_CH2N	TSC_G6_IO4
PB15	SPI2_MOSI	TIM15_CH2	TIM1_CH3N	TIM15_CH1N

GPIO Alternate Function Selection Codes (STM32F051)

Code	Alternate Function
0000	AF0
0001	AF1
0010	AF2

Code	Alternate Function
0011	AF3
0100	AF4 (Port A only)
0101	AF5 (Port A only)
0110	AF6 (Port A only)
0111	AF7 (Port A only)
1000	Reserved
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	Reserved
1111	Reserved

AFR Example

In this example entering “gpioa.” (provided by Svd2forth-v2) produces a listing of all the GPIOA registers, including the MODER and AFR registers, which are reproduced below.

- GPIOA_MODER shows that GPIOA bits 9,10,11 and 12 are configured for *alternate function mode (10)*, while the rest are *input mode (00)*. For convenience, svd2forth-v2 groups the MODER register bit listing in groups of 2 bits.
- GPIOA_AFRL shows that GPIOA bits 0 to 7 are in default *AF0 (0000)*. For convenience, svd2forth-v2 groups the AFRL and AFRH register bit listings in groups of 4 bits.
- GPIOA_AFRH shows that GPIOA bits 9,10,11 and 12 are in *AF1 (0001)*.

```
gpioa.
```

```
GPIOA_MODER $02A80000
15| 14| 13| 12| 11| 10| 09| 08| 07| 06| 05| 04| 03| 02| 01| 00
00 00 00 10 10 10 10 00 00 00 00 00 00 00 00 00

GPIOA_AFRL $00000000
07   06   05   04   03   02   01   00
0000 0000 0000 0000 0000 0000 0000 0000

GPIOA_AFRH $00011110
15   14   13   12   11   10   09   08
0000 0000 0000 0001 0001 0001 0001 0000
```

The reason for this config: This chip is running Mexrisp-Stellaris, which uses USART1 for the serial terminal connection including hardware flow control.

GPIO Pin	Function
9	USART1_TX
10	USART1_RX
11	USART1_CTS
12	USART1_RTS

Antiforth

ANSI Forth is ANTI Forth

Another politically incorrect essay by Geoff Fox. [gefffox]

For years I was a strong advocate for the **ANSIfication of Forth.** Unlike Chuck Moore who just does Forth I had a background from Fox Consulting with clients like Bank of America and Pacific Bell for many years. I mean things couldn't exist without all these standards and people like me love them.

Long before Chuck started laying out my F21 microprocessor my involvement was simulating the chips in development and reporting the results to Chuck and the public. My chip was called the Foxware F20 at the time. I implemented a whole set of different Forth compilers on top of the simulators and published the results. I got tired of the jokes about not having real machines and having to use simulators but I found the research very educational. I published the results of idtc, dtc, stc, ANS and minimal, and optimizing native code compilation with register allocation, inlining, and other cool stuff like parallelization. I guess I could publish more of those ancient documents on the web.

When the iTV Corporation was formed Chuck got together with a couple of guys from NASA and began the mission of making really cheap internet appliances with the technology. They signed a codevelopment deal with Ultra Technology to license an F21 spinoff for their use and they hired me as the first employee.

They hired more ANSI Forth programmers and promoted me from “Cool Software Developer” to “Programming Manager.”

One day Chuck printed out a page on his printer:

Jeff Fox Forth

and came by and put it up on my cube and said, **“I decided to put this up because I decided that that is what you do.”**

In his first public talk about his Color Forth (http://www.ultratechnology.com/color4th.html) Chuck Moore said a couple of things about Forth and in particular ANSI Forth and I know exactly what he was talking about. I have been reluctant to speak out about it because what I will say is so politically incorrect. I don’t have the time to write a book about it right now, but I will write a short web page about why I now say that **ANSI Forth is ANTI Forth.**

Chuck said, **“I had reservations about ANSI. I worried that it would be a disaster and not merely a dubious advantage. All of my fears of the standard and none of the advantages of the standard have come to pass. Any spirit of innovation has been thoroughly quelched. Underground Forths are still needed. I said I thought the standard should be a publication standard but they wanted an execution standard. …**

There is a problem that I first saw at iTV. It was the first time I have been in the presence of a Forth project that I wasn’t contributing to. I saw how other programmers use Forth. They don’t always get it right.

Besides stacks plus colon definitions there is something else. There needs to be another book

written about writing good code."

Neither of us has the time right now to write such a book but I will write this short page with my opinion on the subject of ANSI Forth. I have changed sides once in this debate and would be happy to be convinced that I am wrong about this now.

The **dubious advantage** that Chuck was refering to is related to two illusions. One is that ANSIfication is good because it finally gives Forth credibility in the comp sci and business community. The other is the dangerous illusion of portability in ANSI Forth.

Forth is the simplest language. It is simpler than Logo and it should be taught to children after counting and before long addition. That is the real power of Forth, it is so small and sharp that it can cut up most any problem into well factored testable components.

Forth implementations are usually interactive and offer access to the internals of the compiler and interpreter and everything and make it easy to change anything. Learning to play with those little DUPs and OVERs is like playing with wooden blocks, stacking them and seeing what happens.

Then there is the ANSI Forth specification. It was written by a group of very smart people to benefit the community at large and to cover everyone's behinds regarding how they always implemented things for the last twenty years. It is a well written and clear document for its intents and purposes. But it is not a simple thing like a Forth implementation on a real computer can be. It is a formal description what makes a system compliant with the ANSI standard. It is meant to cover a lot of ground because it is meant to be portable on any type or size of machine or style of implementation.

I would not say that ANSI Forth compilers are not good or that they cannot generate good code. Far from it. I have written lots of them for different companies and they can generate exactly the same object code as some of the other compilers and with source code that is readable to a lot of people.

What I say is that ANSI Forth is just way too big and gives people way way too much room to stay with their bad programming habits from other languages. It certifies and standardizes on lots of bad habits.

At iTV we had programmers who knew ANSI Forth and programmers who didn't know Forth. We had some programmers who wrote exclusively in ANSI Forth, some in either ANSI Forth or Machine Forth, and some who wrote exclusively in Machine Forth. I was programming manager.

Chuck and I constantly argued with management and with the ANSI Forth programmers the following points. Anyone can implement a threaded ANSI certified Forth environment on any chip and only we have the advantage of writing in Machine Forth. We have lots of evidence that code can be written at the same speed in either environment but that it will be four to one hundred times smaller in Machine Forth and ten to a thousand times faster in Machine Forth so we should be doing more of that. We are trying to make a low fat computing product and we are letting a lot of fat get in this way.

One of the programmers would claim that 98% of the bugs that we were working on were simply the result of the illusion of portability and people's insistence that their code works because it was tested in the ANSI Forth development environment that they used. Porting the portable ANSI Forth code from one environment to another found lots of bugs where programmers were just not thinking about how to write portable code. They could not see the bugs in their development environment. Someone else had to port the code. Someone else had to debug the code. The programmers who did this debugging claimed that it was terrible that 98% or our time was chasing down all these bugs related to people's insistance that code was ANSI compliant and tested in their environment. The programmers who did not write code for the target environment would argue that the rest of the programmers were not spending all of their time fixing those kinds of bugs.

I would complain that we were trying to build a low cost, small memory product and that we didn't want to introduce fat. I would also argue that we needed to think of the target as what it was a system optimized for real time demands and minimal resources. When I would find examples of people using standard "portable" ANSI Forth routines from libraries that were 10,000 times slower than the way I would do it and that I could optimize by 100 times in a couple of hours that we needed to think about this portability idea more closely.

My favorite example was the jpeg image decode. A programmer who understood 'C' and ANSI Forth took the task. He translated some jpeg decode code from ANSI 'C' to ANSI Forth and compiled it into the system. It was buggy and the performance was terrible and it was shelved. Another programmer later took on the task. He had not been spoiled by ANSI Forth and knew little about it. He looked at the problem, thought it thru and wrote a little Machine Forth code and debugged it. One program was 100 times smaller and 1000 times faster.

A simple code profile of the various components in the software made it obvious which components were written in Machine Forth and which were written in ANSI Forth. In staff meetings one programmer would report that he got his module down from 3K to 2K and made it 40% faster and another would report that he had only added another 50K to fix his problem and that it was now too slow to measure.

Getting the ANSI Forth programmers to change any of their programming habits was like trying to pull teeth from an angry pit bull by hand. Chuck and I would argue that Machine Forth was so easy and so productive that they should at least try it. They might like it. Chuck and I would argue that Machine Forth limited you to good programming practices and that using ANSI Forth the way they were was making things way more complex than we needed or wanted. Some people got it, some didn’t. Some people would just say that they were hired to write ANSI Forth, and this was the case.

What I see in Forth is a quintessential intellectual toolset.

: FORTH STACKS WORDS BLOCKS ;

What I see as the power in Forth is that it is interactive, real, and simple. You can get right down close to the metal and write the important functions to be as fast as possible and test them in a short amount of time. I really like knowing that when I write DUP that it is a 2ns opcode in our Machine Forth. That lets you write good code. If you don’t know the relative speed of each system word how can your write good code? Well if good means having the potential if carefully written to have some portability at the expense of size and speed then yes ANSI Forth can be good code.

But like Chuck I think that the idea of Forth is to understand and optimize all those little fiddly details to give us a sense of doing the job right. In Machine Forth we have this list of 27 easy to learn and understand Forth opcodes that take 2ns. We took people who never learned ANSI Forth and taught them Machine Forth in an hour and had them programming an embedded wireless ethernet controller in it the next day. ANSI Forth is almost as complex as ‘C’. There are hundreds of words that you have to learn to get started.

People were sensitive about so many people in Forth experimenting and doing things their own way. In the old days Forth was the most portable language because it was the easiest to implement. Then ‘C’ became a standard for portable code. People wanted something like that for Forth so they wanted to get everyone to agree that ANSI Forth was Forth.

I see ANSI Forth as little more than an agreement to officially recognize the pigeon hole where Forth can die. It can become one more failed attempt to find the ideal portable scripting language. I think Chuck sees it the same way. I didn’t see it his way for years but it is one more thing where I have come to agree with him.

I feel insulted when people say that they know ANSI Forth and that they don’t like Chuck using the word “Forth” in the title of his Machine Forth or Color Forth because it is too weird and there is now finally acceptance that

: FORTH dpANSForthspec ;

so he should not call what he is doing Forth any more.

It seems to me that a lot of people think that it isn’t Forth if it isn’t written in ‘C’.

I would like to see $5 computers that are better than PCs. I would like to see Forth enable a new generation of very smart and cheap computers and enable a new generation of very clever children. I see ANSI Forth as an inefficient interactive portable scripting extension to ‘C’ and not at all in the spirit of Forth. I think that that is the disaster of ANSI Forth that Chuck was referring to.

Jeff Fox 2/27/99

I asked Charles Moore to look at this essay and comment.

In email on 3/5/99 he said about this essay:

I like your essays. That figures, since I agree with them. The last months I’ve thought heavily about these issues, with no useful results.

Arrays

This page is a exploration of creating and fetching data from **arrays** using different data types. Hopefully it clearly shows the methods and results.

Arrays in Forth are just indexes to a named base memory location in the Flash memory space. Arrays can only be created in Flash meaning the data must be initialized.

See also

BUFFERS are used for UNINITIALISED data (Ram).

Data Types

Name	Bits	Description
Word	32	Native Mecrisp-Stellaris data length
Cell	32	Same as Word above
Half Word	16	Half length
Character	8	One Byte
Bit	1	Smallest

Note

In Forth $ = 0x or hexadecimal

Emulate c,

Many MCU’s don’t have a native “c,”. Emulate it as below.

```
\ Emulate c, which is not available in hardware on some chips.

0 variable c,collection

: c, ( c -- )
  c,collection @ ?dup if $FF and swap 8 lshift or h,
                      0 c,collection !
                      else $100 or c,collection !
                      then ;

: calign ( -- )
  c,collection @ if 0 c, then ;
```

Create Array of Characters

Four 8 bit or “character” items, $3f, $06, $5b and $66

```
create array $3f c, $06 c, $5b c, $66 c,
```

Fetch the array data using different data types

Fetch Cells

32 bits

```
: =cells       4 * array + dup hex. @  hex.  ;
```

Fetch Half Words

2 bytes or 16 bits

```
: =halfwords   2 * array + dup hex. h@ hex.  ;
```

Fetch Characters

one byte or 8 bits

```
: =characters     array + dup hex. c@ hex.  ;
```

Fetch using different data types

Note the address fetch increments.

Cells

Parameter	Word	Address	Data	Comments
0	=cells	200003A4	665B063F	All the 8 bit data is packed into a CELL, very efficient.
1	=cells	200003A8	2000038C	reading data in memory, not associated with the array
2	=cells	200003AC	3D060000	reading data in memory, not associated with the array

Parameter	Word	Address	Data	Comments
3	=cells	200003B0	6C6C6563	reading data in memory, not associated with the array

Halfwords

Parameter	Word	Address	Data	Comments
0	=halfwords	200003A4	0000063F	
1	=halfwords	200003A6	0000665B	
2	=halfwords	200003A8	0000038C	reading data in memory, not associated with the array

Parameter	Word	Address	Data	Comments
3	=halfwords	200003AA	00002000	reading data in memory, not associated with the array

Characters

Parameter	Word	Address	Data	Comments
0	=characters	200003A4	0000003F	
1	=characters	200003A5	00000006	
2	=characters	200003A6	0000005B	
3	=characters	200003A7	00000066	

Create Array of Halfwords

```
create array $3f4f h, $0607 h, $5b55 h, $6666 h,
```

Fetch the array data using different data types

Fetch Cells

32 bits

```
: =cells       4 * array + dup hex. @  hex.  ;
```

Fetch Half Words

2 bytes or 16 bits

```
: =halfwords   2 * array + dup hex. h@ hex.  ;
```

Fetch Characters

one byte or 8 bits

```
: =characters     array + dup hex. c@ hex.  ;
```

Fetch using different data types

Note the address fetch increments.

Cells

Parameter	Word	Address	Data	Comments
0	=cells	200003A4	06073F4F	
1	=cells	200003A8	66665B55	
2	=cells	200003AC	2000038C	
3	=cells	200003B0	3D060000	

Halfwords

Parameter	Word	Address	Data	Comments
0	=halfwords	200003A4	00003F4F	
1	=halfwords	200003A6	00000607	
2	=halfwords	200003A8	00005B55	
3	=halfwords	200003AA	00006666	

Characters

Parameter	Word	Address	Data	Comments
0	=characters	200003A4	0000004F	
1	=characters	200003A5	0000003F	
2	=characters	200003A6	00000007	
3	=characters	200003A7	00000006	

Create Array of Cells

```
create array $3f4f1234 , $06075678 , $5b558765 , $66664321 ,
```

Fetch the array data using different data types

Fetch Cells

32 bits

```
: =cells       4 * array + dup hex. @  hex.  ;
```

Fetch Half Words

2 bytes or 16 bits

```
: =halfwords   2 * array + dup hex. h@ hex.  ;
```

Fetch Characters

one byte or 8 bits

```
: =characters     array + dup hex. c@ hex.  ;
```

Fetch using different data types

Note the address fetch increments.

Cells

Parameter	Word	Address	Data	Comments
0	=cells	200003A4	3F4F1234	
1	=cells	200003A8	06075678	
2	=cells	200003AC	5B558765	

Parameter	Word	Address	Data	Comments
3	=cells	200003B0	66664321	

Halfwords

Parameter	Word	Address	Data	Comments
0	=halfwords	200003A4	00001234	
1	=halfwords	200003A6	00003F4F	
2	=halfwords	200003A8	00005678	
3	=halfwords	200003AA	00000607	

Characters

Parameter	Word	Address	Data	Comments
0	=characters	200003A4	00000034	
1	=characters	200003A5	00000012	
2	=characters	200003A6	0000004F	

Parameter	Word	Address	Data	Comments
3	=characters	200003A7	0000003F	

Asyncronous Message Passing

Article from Comp Language Forth (Usenet)

Newsgroups: comp.lang.forth Subject: Re: asynchronous message passing From: foxaudioresearch@gmail.com

On Saturday, June 1, 2019 at 10:04:11 AM UTC-4, Dmitry Ponyatov wrote:

> How should I implement asynchronous message passing in FORTH, if I want to implement distributed computing with the actor model? > > I really like the Smalltalk model with messaging at the level of an elementary language operation, and I would like to implement something similar in Forth. > > I feel that it should be done at the level of multiple Forth machines, each of them has its own stack and message dispatch works over the vocabulary. In this model, e

This is very old and simple but it works. Obviously more complex protocols could be added on top of this layer. I have used it in cooperative multi-tasking systems.

As I look at it now it could be simplified with BEGIN WHILE REPEAT .

In this code the word LOCAL computes the task specific address for a user variable.

PAUSE is the task context switch word.

```
\ Based on article in Forth Dimensions Vol 7 #4 by R. W.
Dobbins. Columbia ML.
\ It allows a transputer like com link between FORTH tasks

\ "SENDer" waits until the mailbox is cleared by "RECEIVER"
before
\ sending the next message.
\ Message is 1 CELL.  But complex sender/receiver pairs can
be created with
\ this simple pipe.

: SEND-MSG      ( msg rcvr-PID --)
                 TASK-MSG LOCAL   \ ( msg addr ) compute rcvr
MSGbox address
                 BEGIN
                    PAUSE
                    DUP @          \ ( msg addr n )  n =
contents of MSGbox
                 0= UNTIL         \ ( msg addr ) loop until
MSGbox is empty
                 ! ;                     \ store the msg in
addr

: READ-MSG     ( PID -- msg )     \ read MSG from any task
                 TASK-MSG LOCAL   \ ( -- addr )
                 BEGIN
                   PAUSE
                   DUP @            \ dup the addr and fetch
content
                   ?DUP UNTIL    \ loop until content is not
zero
                   0 ROT ! ;          \ put zero in the MSG
box, msg on stack


: GET-MSG      ( -- n )  myself read-MSG ;
```

Buffers

You are reading a fast 8 bit Analog to Digital Converter and after 10 readings you process the data, but it’s so fast that you only have time to save now and process later, what do you do ?

1. Use the Stack ? You could, but what if you wanted to collect 100 samples ?
2. Use a Buffer. Buffers are used to save/fetch data in (fast) Ram and can be made any length for which you have Ram to spare.

See also

For saving INITIALISED data use an ARRAY

Buffer Creation (Bytes)

This buffer is named “things” and reserves 9 bytes for data. Remember, index 0 is the first and index 8 is the last.

```
compiletoflash
9 buffer: things
compiletoram
```

Buffers: Storing Data

data	buffer	index	add	c store

data	buffer	index	add	c store
$f0	things	0	+	c!
$f1	things	1	+	c!
$f2	things	2	+	c!
$f3	things	3	+	c!

Syntax

```
$f0 things 0 + c!
```

Buffers: Fetching Data

buffer	index	add	c fetch	result
things	0	+	c@	000000F0
things	1	+	c@	000000F1
things	2	+	c@	000000F2
things	3	+	c@	000000F3

Syntax

```
things 0 + c@
```

Practical Usage

A more convenient Word

```
: thing ( # -- a ) things + ;
```

Storing

```
$f2 2 thing  c!
```

Fetching

```
2 thing c@
```

Buffer Creation (Cells)

Cell = 32 bits or byte * 4

```
compiletoflash
10 cells buffer: thingamabobs
compiletoram
```

Create Buffer Access Word

```
: thingamabob ( # -- a ) cells thingamabobs + ;
```

Store to Buffer

```
-1 5 thingamabob !
```

Fetch From Buffer

```
5 thingamabob @
```

The Buffer XT

This shows the Word itself is located in Flash memory space.

```
' things  hex. 0001677E  ok.
```

Yet the “base” index address is in Ram:

```
things hex. 20004E0C  ok.
```

Show Packed Data

```
$20004E0C @ hex. F3F2F1F0  ok.
```

Dump the Buffer

```
$20004E0C dump16 20004E00 :  002A 005E 008A 00A6 0032 00F6
004E 00A1   0070 0041 006B 00DA 00F0 00F1 00F2 00F3  |
*^..2.N.  pAk..... |
```

Proving a buffer only holds X bytes

Thanks to rsoft, mrmobius and rdrop-exit

```
4 buffer: test here test - hex. 00000004  ok.
```

Warning

You may be able to save and retrieve more than 4 bytes in the “test” buffer above, but the data is going somewhere it probably shouldn’t … “buffer overflow” !

Assembly Language: Interactive

- This page shows how to use interactive Assembly Language with Mecrisp-Stellaris via a Blinky example. The reader will need a basic knowledge of Mecrisp-Stellaris Forth and Cortex M Assembly Language.
- I think this is a great way to learn Assembly Language as the Forth Core provides the user support, with recovery only a reset button press on the development board. This is *much faster* than the usual *code, assemble, burn and test* development cycle of Assembly Language which gives no feedback at all on code failure unless additional tools such as OpenOCD and GDB are also used. Even then the only visible evidence may be the Program Counter flying off to the end of memory at the end of the problem code.

See also

Accurate Blocking Delay and Machine and Assembly Code

Environment

- Hardware is a STM32F0 Discovery Board (Cortex M0) but this example should run on any Cortex M supported by Mecrisp-Stellaris.
- Kernel Image is Mecrisp-Stellaris-2.3.9 RA

Software Tools Required

Note: these tools are included with the Mecrisp-Stellaris-2.3.9 release (and many earlier releases) in the mecrisp-stellaris-2.3.9/common directory.

- assembler-m0.txt
    - allows the entry of assembly language into Forth code. Note: It also runs example/test code when loaded to show how it works.
- disassembler-m0.txt
    - provides the ‘see’ Word which ‘disassembles’ Forth Word compiled ‘Machine Code’ in the Dictionary into Assembly Language.

Note

the STM32F0 Discovery Board board doesn’t have enough RAM to load the above files, so compile them into Flash instead.

Registers

Non RA Kernel.

Note

Assembler-m0.txt Opcode Tables can be found at the end of this page

Warning

Never abuse the stack pointer registers for anything else unless you know what you do, or the next interrupt may crash sometimes, which is a nasty bug to find. Stacks always need to be maintained in an interrupt-safe way.

Note

You can use r0, r1, r2 and r3 freely; you need to push/pop r4 and r5 before use, and you can use r6/r7 and r13/r14 as stacks.

Register(s)	Usage Hints
r 0	Saved by IRQ entry – Free scratch register, will be saved automatically on interrupt entry. Do not expect it to stay the same on a subroutine call
r 1	Saved by IRQ entry – Free scratch register, will be saved automatically on interrupt entry. Do not expect it to stay the same on a subroutine call
r 2	Saved by IRQ entry – Free scratch register, will be saved automatically on interrupt entry. Do not expect it to stay the same on a subroutine call

Register(s)	Usage Hints
r 3	Saved by IRQ entry – Free scratch register, will be saved automatically on interrupt entry. Do not expect it to stay the same on a subroutine call
r 4	Is saved by code for every usage, innermost loop index. Save and restore with push/pop if you wish to use
r 5	Is saved by code for every usage, innermost loop limit. Save and restore with push/pop if you wish to use
r 6	Top of stack register. No need to save TOS, as stacks are inherently interrupt safe
r 7	Data stack pointer. No need to save PSP, as stacks are inherently interrupt safe
r 8	Unused, available for your special purposes. Keep in mind that on M0, only a few opcodes are available for them (MOV, ADD, CMP, as far as I know)

Register(s)	**Usage Hints**
r 9	Unused, available for your special purposes. You may get a error/warning ‘Use low registers’
r 10	Unused, available for your special purposes.
r 11	Unused, available for your special purposes.
r 12	Unused, but saved automatically on interrupt entry.
r 13 = SP	Return Stack Pointer, hardwired in silicon
r 14 = LR	Link Register, hardwired in silicon. For bl and bx lr or push { lr} & pop { pc} , automatically saved on interrupt entry
r 15 = PC	Program Counter, hardwired in silicon

The Forth Blinky

- This is a simple Blinky which turns on the blue LED by writing “$100” into memory location $48000814, then turns it off by writing a “0” into the same memory location.

```
$48000800 constant GPIOC                \
GPIOC $0 + constant GPIOC_MODER          \ Hardware Memory
Map
GPIOC $14 + constant GPIOC_ODR            \  GPIOC_ODR =
$48000814, the blue LED is connected to GPIOC-8


%01  16 lshift GPIOC_MODER bis!         \ set PC8 (blue LED)
as output

 : led-on  $100 GPIOC_ODR ! ;           \ turn blue LED on

 : led-off $0   GPIOC_ODR ! ;           \ turn blue LED off

 : delay 900000 0 do loop ;             \ time delay so we
can see the blue LED blinking

 : blink
   begin
      led-on
          delay
      led-off
          delay
   again
 ;


 \ type 'blink' to run the blinky and flash the blue LED,
press the board reset button to stop blinking.
```

Assembly Language ‘asm-led-off’ Word

This Word can replace the ‘led-off’ Word in the Blinky example. Performance of the blinking LED will be identical.

```
: asm-led-off                          \ This is my hand
coded Assembly Language 'asm-led-off' using methods learned
from my recent 'Blinky Challenge' with Matthias
  ldr= r0 $48000800                    \ For clarity I have
chosen the same registers as used by Mecrisp Stellaris when
it compiled 'led-off'
```

```
  ldr= r3 0                             \ Note: assembler-m0.fs must be loaded first, so Assembly Language statements can be used.
  str r3 r0 #$14
;
```

‘asm-led-off’ Disassembled

```
see asm-led-off                       \ 'see' is provided by disassembler-m0.fs
20000402: B500  push { lr }
20000404: 2090  movs r0 #90
20000406: 04C0  lsls r0 r0 #13
20000408: 3080  adds r0 #80
2000040A: 0100  lsls r0 r0 #4
2000040C: 2300  movs r3 #0
2000040E: 6143  str r3 [ r0 #14 ]
20000410: BD00  pop { pc }
ok.
```

‘led-off’ Disassembled

Compare this to ‘asm-led-off’ above, neglecting the ‘push’ and ‘pop’ they’re identical.

```
see led-off
200003CE: 2090  movs r0 #90
200003D0: 04C0  lsls r0 r0 #13
200003D2: 3080  adds r0 #80
200003D4: 0100  lsls r0 r0 #4
200003D6: 2300  movs r3 #0
200003D8: 6143  str r3 [ r0 #14 ]
200003DA: 4770  bx lr
ok.
```

Assembly Language ‘asm-delay’ Word

This Word can replace the ‘delay’ Word in the Blinky example. Performance of the blinking LED will be identical.

```
: asm-delay
     ldr= r0 $fffff
l-:  subs r0 r0 #1
     bne -
;
```

‘asm-delay’ Disassembled

This Word does save *five* instructions compared to the ‘delay’ code generated by Mecrisp-Stellaris, making it a worthwhile exercise.

```
see asm-delay
20000450: B500  push { lr }
20000452: 20FF  movs r0 #FF
20000454: 0200  lsls r0 r0 #8
20000456: 30FF  adds r0 #FF
20000458: 0100  lsls r0 r0 #4
2000045A: 300F  adds r0 #F
2000045C: 1E40  subs r0 r0 #1
2000045E: D1FD  bne 2000045C
20000460: BD00  pop { pc }
```

‘delay’ Disassembled

```
see delay
20000420: B500  push { lr }
20000422: 3F04  subs r7 #4
20000424: 603E  str r6 [ r7 #0 ]
20000426: B430  push { r4  r5 }
20000428: 2400  movs r4 #0
2000042A: 25DB  movs r5 #DB
2000042C: 022D  lsls r5 r5 #8
2000042E: 35BA  adds r5 #BA
20000430: 012D  lsls r5 r5 #4
20000432: CF40  ldmia r7 { r6 }
20000434: 3401  adds r4 #1
20000436: 42AC  cmp r4 r5
20000438: D1FC  bne 20000434
2000043A: BC30  pop { r4  r5 }
2000043C: BD00  pop { pc }
```

Summary

- Handwritten Assembly code can produce smaller code than the Mecrisp-Stellaris generated code, but can also be identical.
- If you'd prefer to use Assembly only with Cortex M, see the Assembly Language article here.

Note

The Forth Handwritten Assembly code above, while smaller by five instructions then the Forth code, actually takes a LOT longer to run probably because the Forth only code is optimised within the Mecrisp-Stellaris system ?

Assembler-m0.txt Opcode Tables

Section Labels

label	Description	Comment
l-	label, backward jump	The assembler resolves the closest 3 labels in every direction relative to the current location. They can be referenced with bne -
l+	label, forward jump	bge ++ bhi - - - which means: one l-: label back, two l+ : labels forward, three l-: back. Different to normal assembler.

Branching

Opcode	Description	Comment
beq	Branch if Z set	equal
bne	Branch if Z clear	not equal
bcs	Branch if C set	unsigned higher or same
bcc	Branch if C clear	unsigned lower
bmi	Branch if N set	negative
bpl	Branch if N clear	positive or zero
bvs	Branch if V set	overflow
bvc	Branch if V clear	no overflow
bhi	Branch if C set and V clear	unsigned higher
bls	Branch if C clear and Z set	unsigned lower or same

Opcode	**Description**	**Comment**
bge	Branch if N set and V set, or N clear and V clear	greater or equal
blt	Branch if N set and V clear, or N clear and V set	less than
bgt	Branch if Z clear, and either N set and V set or N clear and V clear	greater than
ble	Branch if Z set, or N set and V clear, or N clear and V set	less than or equal
bhs	Alias	
blo	Alias	
b	Unconditional branch	

Instructions with its own special handling

Opcode	Description / example	Comment
bl	branch with link, a subroutine call	If range is too far for bl opcode, it will generate a blx r0 sequence.
ldr=	ldr= r0 $48000800	Load register 0 with the Hexadecimal 48000800 (GPIOC BASE address)

Instructions with one register-16 operand

Opcode	Description / example	Comment
b	Unconditional branch	b <label> Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.
blx	Branch and Link (immediate)	calls a subroutine at a PC-relative address

Instructions with two register operands

Opcode	Description / example	Comment
ands		
eors		
adcs		
sbcs		
rors		
tst		
orrs		
muls		
bics		
mvns		
sxtb		

Opcode	Description / example	Comment
sxth		
uxtb		
uxth		
rev		
rev16		
revsh		

Still Under Construction

Opcode	Description / example	Comment
Under Construction !		

Autobaud

Autobaud for a Stm32F0xx Kernel

This Kernel Mod provides Autobaud which syncs to the Terminal after the Enter key is hit. The Baudrate stays in effect until the Target is rebooted or reset.

Tested at 9600, 115200 and 460800 Baud on my STM32F051 Discovery Board which uses a STM32F051 MCU.

Only works on USART1.

Warning

If you normally reset the MCU before uploading new code (as I do), the serial comms will drop out unless you send some CR characters after the reboot or use “forgetram” instead of the “reset” Word if these preceed the new code.

Assembly Source Mods

File to mod is mecrisp-stellaris-X.X.X/mecrisp-stellaris-source/stm32f051-ra/terminal.s

Note

Add the lines that start with a “+” but don’t add the “+” itself. Leave all other code in your STM32F0xx terminal.s alone.

```
@ Enable the USART, TX, and RX circuit
ldr r1, =USART1_CR1
ldr r0, =BIT3+BIT2+BIT0 @ USART_CR1_UE | USART_CR1_TE | USART_CR1_RE
str r0, [r1]

+  .equ USART1_ABREN_Shift, 20   @ bitWidth 1
+  @ set up Autobaud
+  @ The automatic baud rate detection is activated by setting the ABREN bit in the USART_CR2  register.
+  ldr r1, = USART1_CR2
+  movs r2, 0b01
+  lsls r2, USART1_ABREN_Shift
+  str r2, [r1]

bx lr

.include "../common/terminalhooks.s"

@ -----------------------------------------------------------------------------
Wortbirne Flag_visible, "serial-emit"
serial_emit: @ ( c -- ) Emit one character
@ -----------------------------------------------------------------------------
```

Rebuild your Kernel

REBUILD The Kernel

Atomic read/modify accesses

Basically this is the requirement that when a I/O port registers is read or modified, the whole register state is updated before anything else can access it.

Some keywords regarding Atomic operations.

- Unaligned data
- interrupts

Note

NOT all Mecrisp-Stellaris Words are Atomic. For atomic bit set/reset, the ODR bits can be individually set and/or reset by writing to the GPIOx_BSRR or GPIOx_BRR registers

STM32

Some newer STM32 MCU are able to handle I/O port register half-word or byte writes atomically, but older MCU’s such as the STM32F10x cannot and these are disallowed to prevent issues.

STM32F0xx

Each I/O port bit is freely programmable, however the I/O port registers have to be **accessed as 32-bit words, half-words or bytes**. The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIOx_ODR registers. In this way, there is no risk of an IRQ occurring between the read and the modify access.

STM32F10x

As used in the Blue Pill board.

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words, **(half-word or byte accesses are NOT allowed)**. The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.

Yet another reason *not* to use the Blue Pill.

Beginner Advice

Some tips from my own Forth Journey that may be of assistance if you are just starting yours.

Warning

MAJOR PITFALLS

All Forths Are Different

Forth by it’s very nature abhors sameness and cries out to be different. Forth’s creator Charles More believed that “standards” reduced Forth’s ability to solve problems.

The Forth taught in the most famous Forth book of all, “Starting Forth” by Leo Brody has major differences to Mecrisp-Sellaris and others. Many examples won’t work.

This in a way is part of the Forth learning cycle, the “hard yards” a beginner must walk.

If an example from a Forth article won’t work for you, don’t immediately assume that you are at fault, look deeper into why, pop onto the IRC channel or Forum and discuss it. No one will criticise you, we have all been there before.

No Halting on Errors.

Source code is uploaded to the Forth Cortex-M MCU with a serial terminal usually hosted on a PC.

The speed of the serial data is 115200 BPS by default and so the data scrolls up the screen quite quickly. Errors in the source just flow past quietly, no alarm, no colours, no halting of the scrolling and a beginner will probably miss them. All of Forth will be unfamiliar territory to the beginner at that stage so fleeting errors probably won’t catch your eye, they didn’t catch mine.

By default the first indication there has been a error is that the program fails to work, or even worse, the dreaded ‘Unhandled Interrupt 00000003 !’ message scrolls endlessly which pushes your original error message off the end of your terminal buffer so you can’t scroll back to find it after you reset the MCU. This was very frustrating for me.

Fortunately there is a simple remedy thanks to Manfred Mahlow’s e4thCOM terminal. e4thCOM is written in Forth and distributed as a executable with no dependencies, just some minor configuration for the Forth you are using.

Manfred’s e4thCOM terminal will addresses **ALL** the issues above and they will cease to be a problem. Errors will stop the uploads, there will be colour and learning Forth will be fun.

If you use Windows and TerrorTerm, then I feel sorry for you, because you will understandably hate Mecrisp-Stellaris. *Get Linux, embrace the inevitable!*

Lockups

Problem: You’ve just flashed Mecrisp-Stellaris Forth onto a board or chip and your terminal shows the boot up Greetings message.

When you hit the “Enter” key a “OK” response is received and all looks good. However when you upload Forth source code, the terminal locks up after the first few Words requiring the MCU be reset to recover … what’s wrong ??

Answer: The Forth MCU has a compiler inside, and when you enter or upload any source ‘Definitions’ (Words) they are compiled, one after the other.

Now the MCU needs time to compile the Definitions and if new Definitions are uploaded while the chip is still compiling the previous Definition it will appear to ‘lock up’. This is a “Flow Control” issue.

Note

Mecrisp-Stellaris won’t generally ‘lock up’ without giving a error message, of course if you were to write code that disabled the USART, or turned of the CLOCK etc, it may appear that Mecrisp has locked up but this is entirely different to the “Flow Control” problem described above.

Fortunately Manfred Mahlow’s e4thCOM terminal solves this problem as well, the right way.

See also

:ref:Lockups<lockups>

Classic Beginner Mistake

The “blinky’ program loaded fine, no errors, but the LEDS aren’t blinking, what is wrong ???

A: you haven’t RUN the program yet, enter “blinky” and hit the carriage return …

Why won’t my programs work ?

Lots of forth programs may not work for you, even Mecrisp-Stellaris programs but don’t be disheartened, there are many reasons and they can all be fixed.

Definitions

Missing Definitions

The program will be expecting certain Words that are not in your Dictionary, they are dependencies and should have been put there already. Manfred Mahlow’s e4thCOM terminal can fix this problem also.

Wrong Definitions

The definition may have the wrong information, be for another type of chip, variations are many, fix them one at a time.

Stack problems

Leaving data on the Stack is a major issue for beginners and cause all kind of weird problems, including the dreaded ‘Unhandled Interrupt 00000003 !’.

Watch your Stack constantly with “.s”, check it after every Word. Make sure that if nothing is supposed to be left on the Stack, that nothing is.

Now you can have a COLOURED PROMPT to show a non empty stack!

Long Words

We all write Words that are *way too long* when we start learning Forth because :

1. If we came from other programming languages then long programs are common and not a problem.
2. We erroneously believe that more Words uses more memory and is less efficient. It isn’t. Forth is designed to use lots of short Words for many good reasons. Aim for three short lines of code per Word.
3. Writing short Words is *HARD*, it takes Zen Forth Master status! It’s easy to write a 50 line Word, but not so easy to debug or understand it however.

Benchmarks

The question of **Speed** often crops up, so I’ve listed some comparisons of programming languages and MCU’s below.

Programming Language Benchmark

Computing the same 100-term polynomial 500,000 times, smaller is faster.

I’ve borrowed the table below from http://dan.corlan.net/bench.html which you will need to read for the full story.

Note

Performed on a 300MHz Pentium using Debian GNU/Linux, so it’s fairly old

Language	single body (s)	with call (s)
FORTRAN, g77 V2.95.4	2.73	2.73
Ada 95, gnat V3.13p	2.73	2.74
C, hand optimized , gcc V2.95.4	2.73	

Language	**single body (s)**	**with call (s)**
Java, gcj V3.0	3.03	15.53
D, gcc V4.0.3+	1)3.43	1)3.98
C, gcc V2.95.4	3.61	3.57
R translated to lisp using R2cl v0.1 and compiled with cmucl	3.69	
Lisp, CMU Common Lisp V3.0.8 18c+ , build 3030	4.69	10.69
Java, jikes V1.15 (bytecompiled)	8.23	13.54
FORTH, hand optimized Gforth 0.6.1	1)18.21	
FORTH,** Gforth 0.6.1	1)27.26	
Python** + psyco (interpreted)	1)168.50	
Perl, more optimized$ V5.6.1 (natively compiled)	209.20	

Language	**single body (s)**	**with call (s)**
Perl, more optimized$ V5.6.1 (interpreted)	258.64	
Perl, hand optimized*** V5.6.1 (bytecompiled)	306.18	
Perl* V5.6.1 (natively compiled)	367.23	
Python** V2.1.2 (interpreted)	505.50	
Perl* V5.6.1 (bytecompiled)	515.04	
RUBY*** (interpreted)	1074.52	
R V1.5.1 (interpreted)	5662.64	

Power Of A Language Or Performace Of An Implementation ?

This is the big question to me:

- What if the project itself doesn’t need the maximum processing *speed* of the target MCU, but rather requires the *minimum development time* ? I prefer the interactivity of Forth so I can quickly determine hardware characteristics when beginning a new project. This saves me a lot of time correcting invalid assumptions in my code later on.

Benchmarking Different Forths/Mcus

Calculate the greatest common divisor for 0 to 200 **`Download: GCD`**

Inspired by : http://weblambdazero.blogspot.com.au/search/label/forth

Benchmark: Calculate The Greatest Common Divisor For 0 to 200

Less time is best, ordered by fastest at top.

MCU	Clock (MHz)	Time (sec)	Comments
STM32F103	75	0.1505	Mecrisp-Stellaris RA 2.5.1 with M3 core for STM32F103
STM32F051	96 (overclocked)	0.3	Mecrisp-Stellaris RA 2.3.7 with M0 core for STM32F051

MCU	Clock (MHz)	Time (sec)	Comments
STM32F051	72 (overclocked)	0.516	Mecrisp-Stellaris RA 2.5.3 with M0 core for STM32F051
STM32F051	48	0.6	Mecrisp-Stellaris RA 2.3.7 with M0 core for STM32F051
STM32F051	8	2.59	Mecrisp-Stellaris RA 2.3.7 with M0 core for STM32F051
Atmega328	16	4	Flash Forth 5
STM8S003F3P6 (From Flash)	16	6.4	optimized for size not speed
STM8S003F3P6 (From Ram)	16	6.9	optimized for size not speed

Atmega328

Forth	**Clock**	**Time (Sec)**
Flash Forth 5:	16	4
AmForth 6.3	16	8
Yaffa Forth 0.6.1	16	70

Code

```
: gcd ( a b -- gcd )                \  15 25 gcd .
  begin
   dup
   while
     swap over mod
  repeat
  drop ;

: bench ( n -- )
  ms.counter.reset
  200 dup 0 do
     dup 0 do
       j i gcd
       drop
     loop
  loop drop
  ms.print cr
;

bench
```

BITFIELDS AND STRUCTURES

When I was writing keil c v/s svd2forth I wondered why so many embedded C Programming Languages use C Structures as their way of dealing with Register Bitfields. This page looks into that preference and compares it to the way Forth might deal with the same issues.

It has been suggested that *“You often need to make sure the right accesses are being generated by the C compiler, if they’re defined as 32/64-bit integers in a struct you can be sure changes will generate the correct sized store/move. In Forth the access size is very explicit everywhere so it’s not ‘hidden’ in something like that. In C it tries to hide it… and then every embedded C programmer has to turf it up somehow.”*

WORK
IN PROGRESS

BITFIELDS, DECLARE EARLY ?

This article is an addition to my bitfield series:

1. keil c v/s svd2forth
2. Bitfields and Structures

Why I Use and Declare Bitfields Early

I believe there are two Forth styles:

1. Programmer Style

This style deals with Forth programming, is efficient, well written and easily understood. It’s everything a Forth user appreciates.

```
\ Memmap
$40002800 constant RTC
RTC $C + CONSTANT RTC_ISR
RTC $10 + CONSTANT RTC_PRER

\ Program Code
: rtc-cal-set-on
$80 RTC_ISR bis!
BEGIN $40 RTC_ISR BIT@ UNTIL    \ Calendar registers update is allowed.
$7f00ff RTC_PRER !              \ From LSI, LSE - $ff
$7f00ff RTC_PRER !
;
```

2. My Electronics Technician Style

My style (below) looks like it was written by a bored programmer on Friday afternoon. It’s longer than the code above and most programmers would want to factor out Bitfields below as much of it seems redundant to them. The Forth programmer mind simply cannot resist factoring everything into its most basic form.

And that’s a very good thing, a highly honed efficiency skill that is somewhat rare in a world of excess computing resources.

Then why do I use *bitfields*, what information do they provide that the *Programmer Style* above does not ?

The answer for the *impatient* is **HARDWARE INFORMATION** which it totally lacking in the Programmer Style above. Please permit me to explain below.

```
\ Memmap
$40002800 constant RTC
RTC $C + CONSTANT RTC_ISR
RTC $10 + CONSTANT RTC_PRER


\ Bitfields
\ RTC_ISR () Reset:0x00000007
: RTC_ISR_INIT ( -- x addr ) 7 bit RTC_ISR ;
\ RTC_ISR_INIT, Initialization mode
: RTC_ISR_INITF? ( -- x addr ) 6 bit RTC_ISR bit@ ;
\ RTC_ISR_INITF, Initialization flag

\ RTC_PRER (read-write) Reset:0x007F00FF
: RTC_PRER_PREDIV_A ( %bbbbbbb -- x addr ) 16 lshift RTC_PRER
;  \ RTC_PRER_PREDIV_A, Asynchronous prescaler  factor
: RTC_PRER_PREDIV_S ( %bbbbbbbbbbbbbbb -- x addr ) RTC_PRER ;
\ RTC_PRER_PREDIV_S, Synchronous prescaler  factor


\ Program Code
: rtc-cal-set-on
  RTC_ISR_INIT BIS!
```

```
\ Enter RTC Initialization mode
  begin RTC_ISR_INITF? until
\ Loop until Calendar has been initialized
  %1111111 RTC_PRER_PREDIV_A bis!
\ Asynchronous prescaler factor
  %11111111 RTC_PRER_PREDIV_S bis!
\ Synchronous prescaler factor
;
```

Programmer Style Example

In this example I use source from the Mecrisp-Stellaris user contributions to deconstruct a code segment and analyse the configuration of *hardware* involved.

We can’t see how it works without referring to the **`Technical Reference`** for the MCU involved. This can be very time consuming especially if has to be done for every peripheral register used in the source.

```
$40002800 constant RTC
RTC $C + CONSTANT RTC_ISR
RTC $10 + CONSTANT RTC_PRER

: rtc-cal-set-on
$80 RTC_ISR bis!
BEGIN $40 RTC_ISR BIT@ UNTIL \ Calendar registers update is
allowed.
$7f00ff RTC_PRER !           \ From LSI, LSE - $ff
$7f00ff RTC_PRER !
;
```

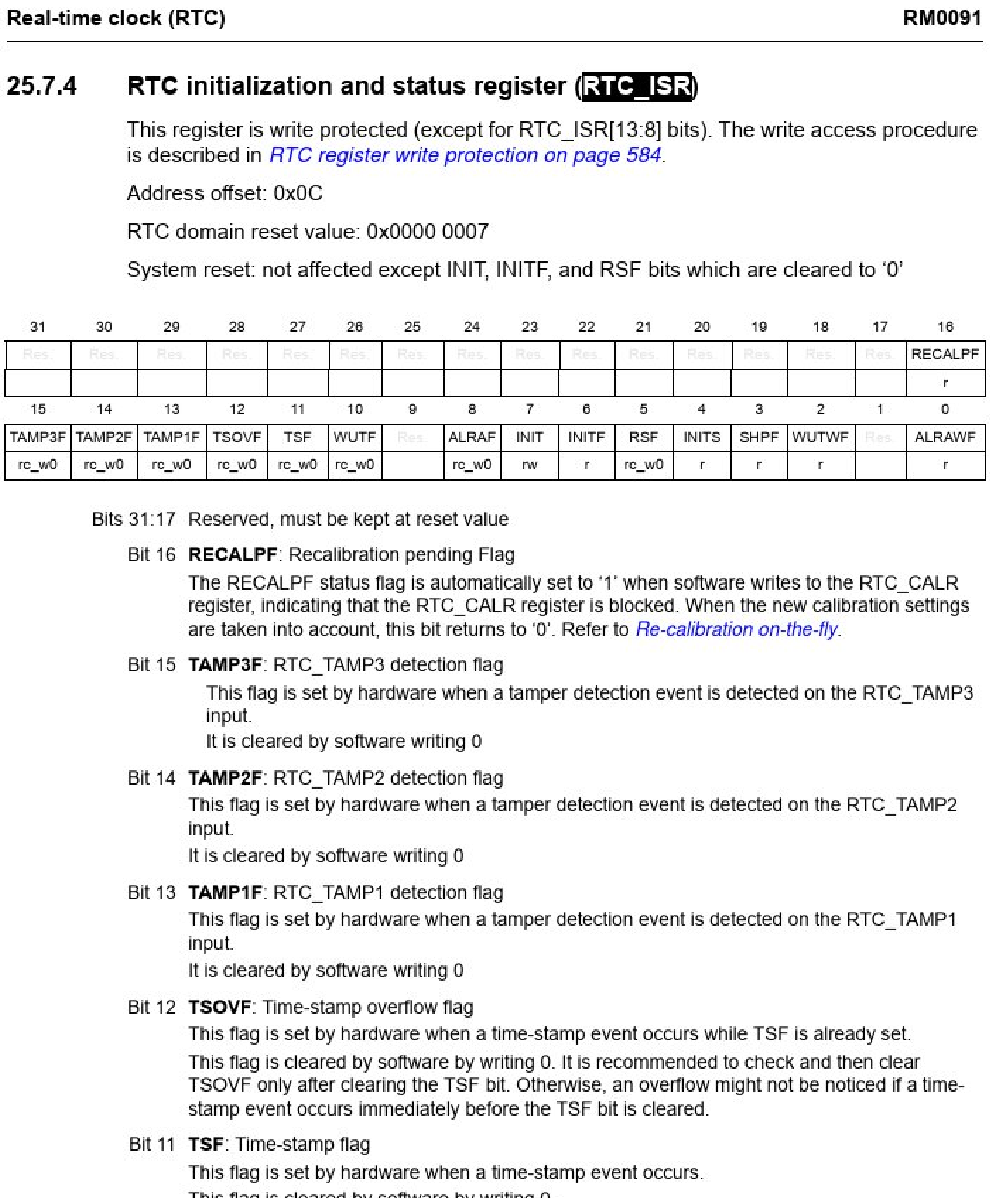
Deconstructing the Programmer Style Example

$80 RTC_ISR bis!

This means “set bit 7 of RTC_ISR”.

```
$80 bin. $00000080
3322222222221111111111
10987654321098765432109876543210
00000000000000000000000010000000
```

But *what IS bit7* ? We need to look in the Technical manual to find that out that *bit7* is the RTC *init mode*



Real-time clock (RTC) RM0091

25.7.4 RTC initialization and status register (RTC_ISR)

This register is write protected (except for RTC_ISR[13:8] bits). The write access procedure is described in *RTC register write protection on page 584*.

Address offset: 0x0C

RTC domain reset value: 0x0000 0007

System reset: not affected except INIT, INITF, and RSF bits which are cleared to ‘0’

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RECALPF
															r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAMP3F	TAMP2F	TAMP1F	TSOVF	TSF	WUTF	Res.	ALRAF	INIT	INITF	RSF	INITS	SHPF	WUTWF	Res.	ALRAWF
rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0		rc_w0	rw	r	rc_w0	r	r	r		r

Bits 31:17 Reserved, must be kept at reset value

Bit 16 **RECALPF**: Recalibration pending Flag

The RECALPF status flag is automatically set to ‘1’ when software writes to the RTC_CALR register, indicating that the RTC_CALR register is blocked. When the new calibration settings are taken into account, this bit returns to ‘0’. Refer to *Re-calibration on-the-fly*.

Bit 15 **TAMP3F**: RTC_TAMP3 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP3 input.

It is cleared by software writing 0

Bit 14 **TAMP2F**: RTC_TAMP2 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP2 input.

It is cleared by software writing 0

Bit 13 **TAMP1F**: RTC_TAMP1 detection flag

This flag is set by hardware when a tamper detection event is detected on the RTC_TAMP1 input.

It is cleared by software writing 0

Bit 12 **TSOVF**: Time-stamp overflow flag

This flag is set by hardware when a time-stamp event occurs while TSF is already set.

This flag is cleared by software by writing 0. It is recommended to check and then clear TSOVF only after clearing the TSF bit. Otherwise, an overflow might not be noticed if a time-stamp event occurs immediately before the TSF bit is cleared.

Bit 11 **TSF**: Time-stamp flag

This flag is set by hardware when a time-stamp event occurs.

This flag is cleared by software by writing 0.

This flag is cleared by software by writing 0.

Bit 10 **WUTF**: Wakeup timer flag

This flag is set by hardware when the wakeup auto-reload counter reaches 0.

This flag is cleared by software by writing 0.

This flag must be cleared by software at least 1.5 RTCCLK periods before WUTF is set to 1 again.

Bit 9 Reserved, must be kept at reset value.

Bit 8 **ALRAF**: Alarm A flag

This flag is set by hardware when the time/date registers (RTC_TR and RTC_DR) match the Alarm A register (RTC_ALRMAR).

This flag is cleared by software by writing 0.

Bit 7 **INIT**: Initialization mode

0: Free running mode
1: Initialization mode used to program time and date register (RTC_TR and RTC_DR), and prescaler register (RTC_PRER). Counters are stopped and start counting from the new value when INIT is reset.

Bit 6 **INITF**: Initialization flag

When this bit is set to 1, the RTC is in initialization state, and the time, date and prescaler registers can be updated.

0: Calendar registers update is not allowed
1: Calendar registers update is allowed

Bit 5 **RSF**: Registers synchronization flag

This bit is set by hardware each time the calendar registers are copied into the shadow registers (RTC_SSRx, RTC_TRx and RTC_DRx). This bit is cleared by hardware in initialization mode, while a shift operation is pending (SHPF=1), or when in bypass shadow register mode (BYPSHAD=1). This bit can also be cleared by software.

It is cleared either by software or by hardware in initialization mode.

0: Calendar shadow registers not yet synchronized
1: Calendar shadow registers synchronized

Bit 4 **INITS**: Initialization status flag

This bit is set by hardware when the calendar year field is different from 0 (RTC domain reset state).

0: Calendar has not been initialized
1: Calendar has been initialized

Bit 3 **SHPF**: Shift operation pending

0: No shift operation is pending
1: A shift operation is pending

This flag is set by hardware as soon as a shift operation is initiated by a write to the RTC_SHIFTR register. It is cleared by hardware when the corresponding shift operation has been executed. Writing to the SHPF bit has no effect.

Bit 2 **WUTWF**: Wakeup timer write flag

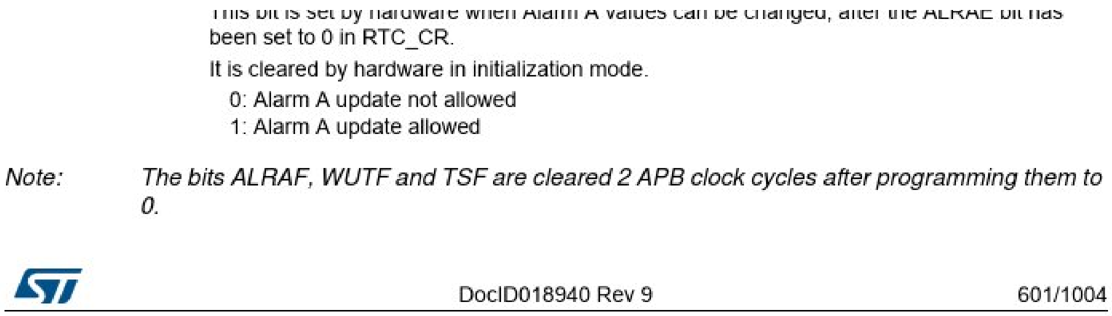
This bit is set by hardware up to 2 RTCCLK cycles after the WUTE bit has been set to 0 in RTC_CR, and is cleared up to 2 RTCCLK cycles after the WUTE bit has been set to 1. The wakeup timer values can be changed when WUTE bit is cleared and WUTWF is set.

0: Wakeup timer configuration update not allowed
1: Wakeup timer configuration update allowed

Bit 1 Reserved, must be kept at reset value.

Bit 0 **ALRAWF**: Alarm A write flag

This bit is set by hardware when Alarm A values can be changed, after the ALRAE bit has


This bit is set by hardware when Alarm A values can be changed, after the ALRAE bit has been set to 0 in RTC_CR.

It is cleared by hardware in initialization mode.

0: Alarm A update not allowed
1: Alarm A update allowed

Note: The bits ALRAF, WUTF and TSF are cleared 2 APB clock cycles after programming them to 0.

DocID018940 Rev 9 601/1004

BEGIN $40 RTC_ISR BIT@ UNTIL

This code reads bit 6 (INITF) of the RTC_ISR, looping until it is HIGH

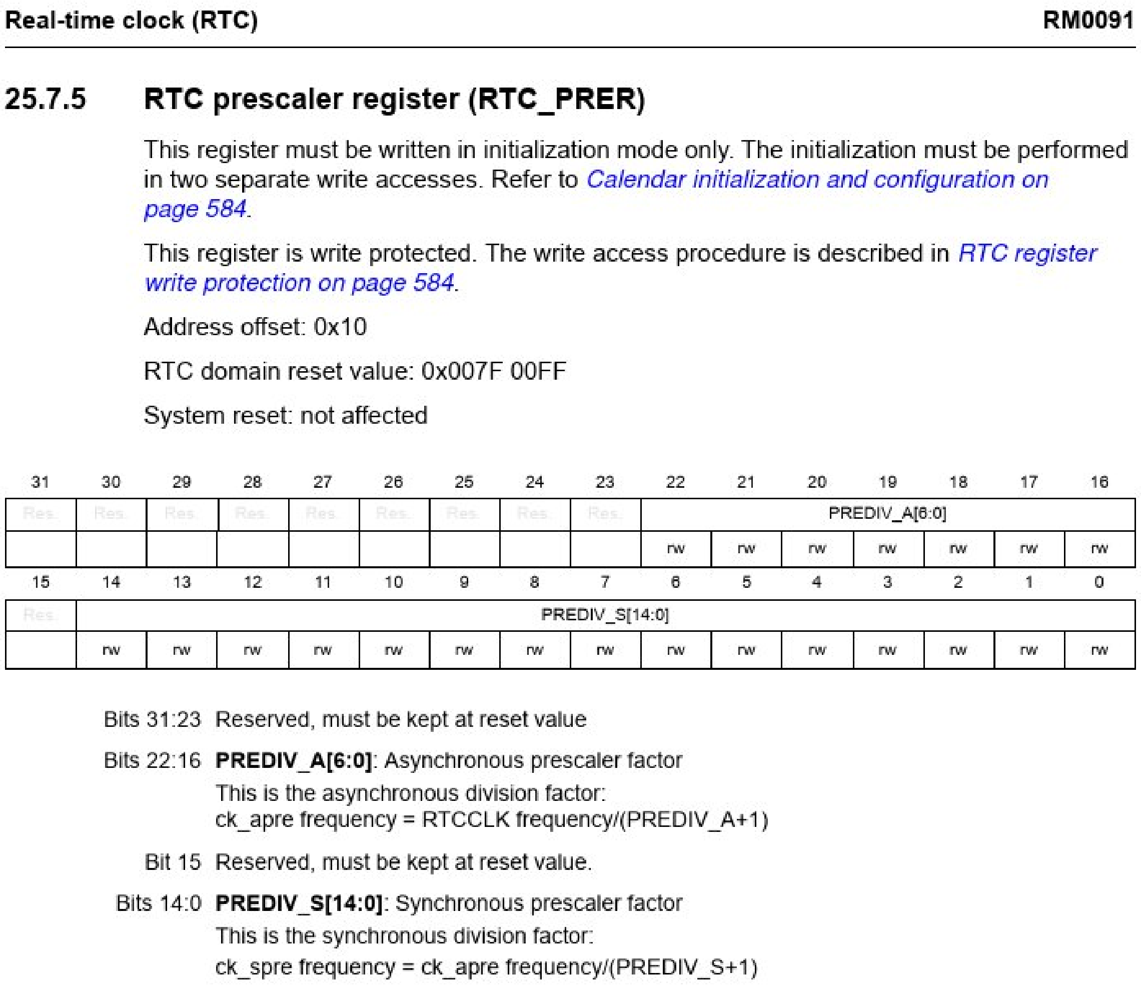
```
$40 bin. $00000040
3322222222221111111111
10987654321098765432109876543210
00000000000000000000000001000000
```

$7f00ff RTC_PRER !

Writes $7f00ff to the RTC_PRER *register*. But why *twice* ?

```
$7f00ff RTC_PRER !          \ From LSI, LSE - $ff
$7f00ff RTC_PRER !
```

RTC_PRER Register



Real-time clock (RTC) RM0091

25.7.5 RTC prescaler register (RTC_PRER)

This register must be written in initialization mode only. The initialization must be performed in two separate write accesses. Refer to *Calendar initialization and configuration on page 584*.

This register is write protected. The write access procedure is described in *RTC register write protection on page 584*.

Address offset: 0x10

RTC domain reset value: 0x007F 00FF

System reset: not affected

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PREDIV_A[6:0]						
									rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PREDIV_S[14:0]														
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23	Reserved, must be kept at reset value
Bits 22:16	**PREDIV_A[6:0]**: Asynchronous prescaler factor This is the asynchronous division factor: ck_apre frequency = RTCCLK frequency/(PREDIV_A+1)
Bit 15	Reserved, must be kept at reset value.
Bits 14:0	**PREDIV_S[14:0]**: Synchronous prescaler factor This is the synchronous division factor: ck_spre frequency = ck_apre frequency/(PREDIV_S+1)

The RTC_PRER register has two bitfields, “RTC_PRER_PREDIV_A” and “RTC_PRER_PREDIV_S”.

It’s obvious the coder is writing both the RTC_PRER_PREDIV_A and RTC_PRER_PREDIV_S bitfields at the same time by writing to the RTC_PRER *register* with a CELL wide value. But why is this done TWICE ? It’s because the RTC prescaler register (RTC_PRER) initialization must be performed in two separate write accesses, i.e. once for RTC_PRER_PREDIV_A and once for RTC_PRER_PREDIV_S

Looking at the binary value being written into register RTC_PRER. One can see that the $7F part will be written to

“RTC_PRER_PREDIV_A” and the $FF part to “RTC_PRER_PREDIV_S” at the same time. This is then done again to complete the required sequence.

```
$7f00ff RTC_PRER !          \ From LSI, LSE - $ff

$7f00ff bin. $007F00FF
3322222222221111111111
10987654321098765432109876543210
00000000011111110000000011111111
```

Electronics Technician Style

Otherwise known as “Declare Bitfields Early”, this style uses bitfield definitions derived from the CMSIS-SVD file released by the manufacturer of the MCU and transformed by XSLT.

The Bitfields paragraph contains vital hardware information, necessary for the easy maintenance of this code years later and possibly negating the requirement of having a **`Technical Reference`** on hand ?

The bitfield definitions tells me whether a bitfield is a single bit that requires setting, clearing or testing or whether it is a multi bit choice and how many bits there are. The comments field also provides a terse summary of the bitfield function.

Finally the name of the bitfield definition is CMSIS-SVD compliant and can be used to search a technical reference PDF for more information.

```
\ Memmap
$40002800 constant RTC
RTC $C + CONSTANT RTC_ISR
RTC $10 + CONSTANT RTC_PRER



\ Bitfields
\ RTC_ISR () Reset:0x00000007
```

```
: RTC_ISR_INIT ( -- x addr ) 7 bit RTC_ISR ;
\ RTC_ISR_INIT, Initialization mode
: RTC_ISR_INITF? ( -- x addr ) 6 bit RTC_ISR bit@ ;
\ RTC_ISR_INITF, Initialization flag
\ RTC_PRER (read-write) Reset:0x007F00FF
: RTC_PRER_PREDIV_A ( %bbbbbbb -- x addr ) 16 lshift RTC_PRER
;  \ RTC_PRER_PREDIV_A, Asynchronous prescaler  factor
: RTC_PRER_PREDIV_S ( %bbbbbbbbbbbbbbb -- x addr ) RTC_PRER ;
\ RTC_PRER_PREDIV_S, Synchronous prescaler  factor


\ Program Code
: rtc-cal-set-on
  RTC_ISR_INIT BIS!
\ Enter RTC Initialization mode
  begin RTC_ISR_INITF? until
\ Loop until Calendar has been initialized
  %1111111 RTC_PRER_PREDIV_A bis!
\ Asynchronous prescaler factor
  %11111111 RTC_PRER_PREDIV_S bis!
\ Synchronous prescaler factor
;
```

Summary

Embedded programmers have to worry about the hardware *and* the software, they are of equal importance, now when the program is written, and years later when it’s maintained, upgraded or just repaired.

Removing all the hardware information isn’t *embedded programming*, it’s just *programming*.

Bit Manipulation Tips

With examples

XOR Bit Resetting

Backstory

GPIOA PIN 12 had to be changed from the default “ANALOG” Mode, to “ALTERNATE FUNCTION” Mode. The MCU was a STM32L073, which is a Cortex-M0+ .

GPIO PINs are configured within the MODER register in pairs as can be seen from the table below.”%” in Forth means “binary value”

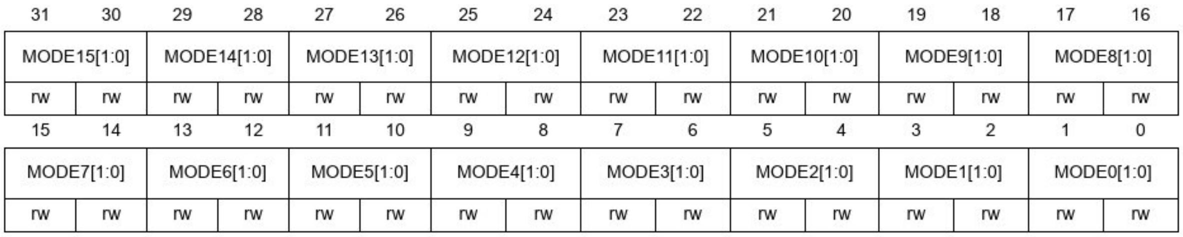
The existing GPIOA_MODER value was $EBEBFCFF and all other bits had to remain unchanged..

This is the binary value of $EBEBFCFF, (the value of GPIOA_MODER) and the bits are grouped in pairs for clarity. PIN 12 consists of bits 24 and 25.

```
GPIOPA_MODER: $EBEBFCFF
15| 14| 13| 12| 11| 10| 09| 08| 07| 06| 05| 04| 03| 02| 01| 00
11 10 10 11 11 10 10 11 11 11 11 00 11 11 11 11
```

By default, all pins after power reset are configured to “ANALOG” and bit 12 was still default. To change it to AF (ALTERNATE FUNCTION) it had to be changed from “%11” to “10”. This was done as below.

GPIOx_MODER Table



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

MODER	Mode
%00	INPUT
%01	OUTPUT
%10	AF
%11	ANALOG

Operation

Note

All commands in CAPITALS are standard Mecrisp-Stellaris Words.

```
----------------------------- -------------------------------
-------------------------------- ---------
Operation                     Binary Value
Hex Value
----------------------------- -------------------------------
-------------------------------- ---------
Legend
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
Legend
```

```
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
$EBEBFCFF                     1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1
1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 $EBEBFCFF
1 24 LSHIFT                   0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 $01000000
$EBEBFCFF $01000000 XOR       1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1
1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 $EAEBFCFF
```

Summary

After the operation, the new value was $EAEBFCFF and this was used as a value in a Assembly Language program.

```
@ Set PORTA TX, RX and RTS in alternate function mode
ldr r1, = GPIOA_MODER
ldr r0, = 0xEAEBFCFF  @ EBFFFCFF is reset value for Port A
str r0, [r1]
```

Bugs or Benefits ?

Unhandled interrupt 0000000X !

Printed over and over again in the terminal when you power up the board.

One possible reason is that the Flash memory contained other code when you loaded Mecrisp-Stellaris, and the flash method you used ‘automatically’ erased enough Flash memory for Mecrisp-Stellaris, but didn’t erase the previous code area. What needs to be done is a block erase od ALL FLASH before loading Mecrisp-Stellaris.

Old Words Remain After Upgrading Mecrisp-Stellaris!

I’ve just flashed a new version of Mecrisp-Stellaris and my old Words are still there !

Is this a bug or a feature ? Actually it has nothing to do with Mecrisp-Stellaris and the answer is exactly the same as in Q1) above. If you want to completely remove your old Words you need to do a block erase of the board Flash memory.

Unhandled interrupt 00000003

My new Word results in endless “Unhandled interrupt 00000003” messages and needs hard a reset to stop it.

interrupt 00000003 is a ‘catch all’ for any EXCEPTIONS that have been raised but not handled. A likely cause is your program caused the Program Counter to stray somewhere it’s not supposed to.

Q: How can I stop that “Unhandled Interrupt 00000003 !” message ?

If I for example read beyond RAM from $20008000 I would expect that it would report once and then carry on as normal or at least reset afterwards. At present I have to reset it from the debugger after it has scrolled up thousands of lines off the terminal.

Answer:

You can change the way fault interrupts are handled by hooking “irq-fault” which points to “unhandled” for default.

For example, load the calltrace utility (mecrisp-stellaris-2.5.4/common/calltrace.txt) and install its handler:

```
' ct-irq irq-fault !
```

This one traps on faults; you can replace “begin again” at the end of ct-irq with “reset” if you like.

A few notes and example of the calltrace tool

You can tap any interrupt by writing a (–) definition, tick it and place its address into any of the irq-* pointer variables. Forth will handle all housekeeping necessary for interrupt entry and return. The interrupts which do not have individual hooks launch through “irq-common”. Inside of your collection handler you can do something like “ipsr case”.

Fault interrupts are always active, for others, you have “eint” “dint” and “eint?” at your fingertips.

One more note: If you compile a definition called “init” into flash, it will be executed automatically on boot.

```
compiletoflash
load calltrace.txt
init ( -- ) ['] ct-irq irq-fault ! ;
```

This will install the calltrace utility permanently.

Matthias

Bluepill Examples

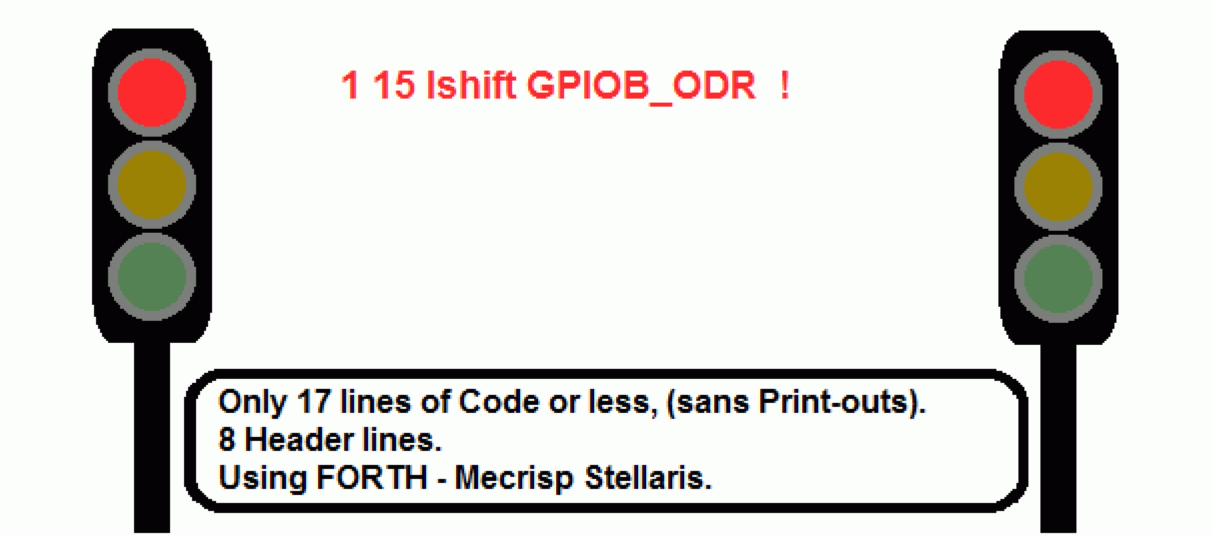
User contributed example programs for the cheap chinese BLUE PILL board

See also

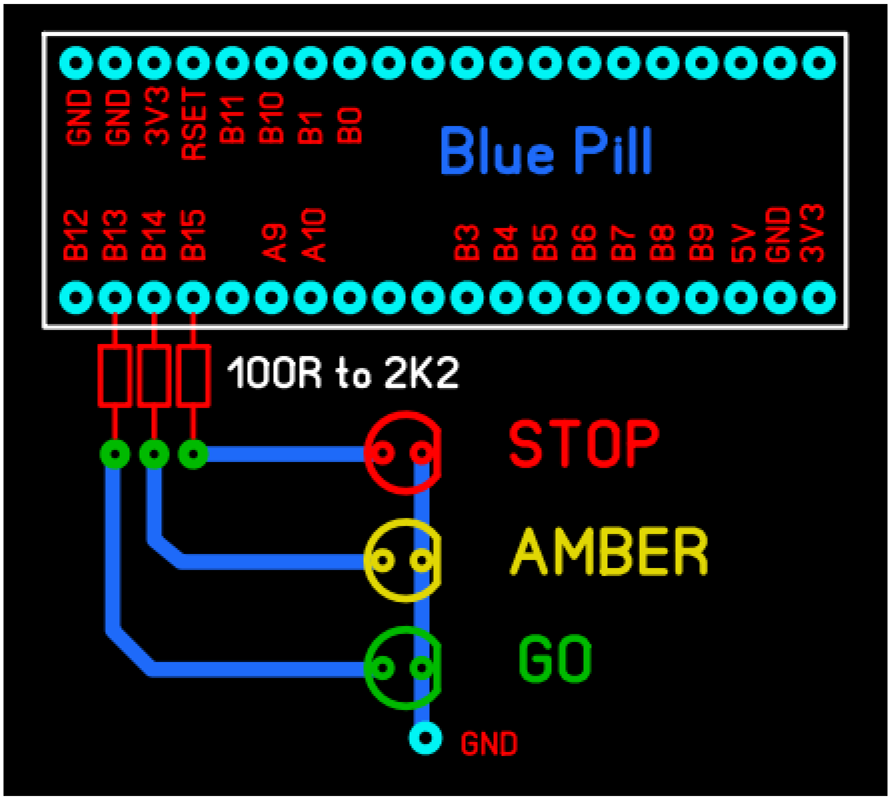
The Blue Pill starter kit and my anti Blue Pill rant *especially if you have not yet bought one* ?

Traffic Lights

By Michael Jeffery.


1 15 lshift GPIOB_ODR !

Only 17 lines of Code or less, (sans Print-outs).
8 Header lines.
Using FORTH - Mecrisp Stellaris.


Blue Pill
GND
GND
3V3
RSET
B11
B10
B1
B0
B12
B13
B14
B15
A9
A10
B3
B4
B5
B6
B7
B8
B9
5V
GND
3V3
100R to 2K2
STOP
AMBER
GO
GND

DOWNLOAD traffic.txt

```
\ Traffic Light, By Michael Jeffery Copyright 2019, licensed under the GPL.
\ For use with Mecrisp-Stellaris by Matthias Koch. https://sourceforge.net/projects/mecrisp/
\ The original shell of this was taken from the Matthias Koch 'Blinky' and reworked.
\ Thanks also to Terry Porter.
\ Traffic Light, uses PB13, PB14, PB15 on a Blue Pill Module
running Forth - Mecrisp Stellaris.
\ This will run a Traffic Light sequence, around 12 sec for
each of the Green and Red LEDs.
\ You can change the Timing sequence in the Loop, 25000000 0
or 4200000 0 for a different result.
\ In Terminal you should see the word 'STOP' then 'AMBER'
then 'GO' printed with four or so spaces
\ after each word on every new step.
\ As the delay loops are Long, 'key? until' only works at the
end of each cycle, so you may have
\ to wait a little for press 'any key' to end the program. An
interrupt would work better here.
\ Copy and paste this text into your Terminal, hit enter,
then type Traffic when it's done.
\ If you get errors, set the EOL (End Of Line) delay to 200ms
in your Terminal.
\ Type Traffic to Start the sequence.


$40010C00 constant GPIOB

GPIOB $00 + constant GPIOB_CRL \ Reset value: $44444444 PB0-
PB7 are configured in CRL, PB8-PB15 in CRH.
GPIOB $04 + constant GPIOB_CRH \ Reset value: $44444444 1:
Push-Pull output, 4: Floating input.
GPIOB $08 + constant GPIOB_IDR
GPIOB $0C + constant GPIOB_ODR         \ Configuration
Settings for GPIO Port B
GPIOB $10 + constant GPIOB_BSRR        \ on the Blue Pill
Module.
GPIOB $14 + constant GPIOB_BRR
GPIOB $18 + constant GPIOB_LCKR

: Traffic ( -- )          \ Start of Traffic 'Word'.
$11144444 GPIOB_CRH !     \ PB15: PB14: PB13: sets Push-Pull
Output.
begin                     \ Start Begin - Until loop.
1 13 lshift GPIOB_ODR !   \ GREEN, GO ON, Set Pin13 ON.
." GO    "                \ Prints GO on the Terminal.
25000000 0 do loop        \ Delay Loop, XXX to 0.
1 14 lshift GPIOB_ODR !   \ AMBER, Wait ON, Set Pin14 ON.
." AMBER    "             \ Prints AMBER on the Terminal.
4200000 0 do loop         \ Delay Loop, XXX to 0.
```

```
1 15 lshift GPIOB_ODR !   \ RED, STOP ON, Set Pin15 ON.
." STOP    "              \ Prints STOP on the Terminal.
25000000 0 do loop        \ Delay Loop, XXX to 0.
1 14 lshift GPIOB_ODR !   \ AMBER, Wait ON, Set Pin14 ON.
." AMBER    "             \ Prints AMBER on the Terminal.
4200000 0 do loop         \ Delay Loop, XXX to 0.
key? until                \ Ends if any Key pressed, or Loops
forever.
;                         \ End of traffic 'Word'.



\ So here we created a single new Word called Traffic, that's
basically how Forth works.
\ Every time you type this Word the Code will now run (as
long as you save it, see below).
\ You will need to include the Header first to configure
GPIOB as well, if not previously saved.


\ If you are in the UK you may want to leave everything 'as
is'.
\ If you are in the USA you may want to remove the last AMBER
set, so Green goes directly to
\ Red. There is no Red and Amber at once before Green though,
that's the advanced version ~ :o)
\ If you want this Word 'Traffic' to remain after a reset you
could include 'compiletoflash'
\ above the Header, without the quotation marks of course and
it will then be included
\ in your dictionary, along with the Header config for GPIOB.


\ Wiring, any LED Resistor value from around 120 Ohms to 2K2
Ohms should work, LED Pos+ to the
\ Blue Pill pin PB13, PB14 and PB15. Blue Pill output 'high'
is 3.3V.
\ If in doubt use an online LED / Resistor Calculator,
there's hundreds out there.
\ Other than that you probably won't get any further support
on this part of the project.
\
\ Good Luck!
\ Swap Red for Green order, starts on RED vs Green, easy just
swap each LED part.
\ Use the same Header as above also.

\ : Traffic ( -- )
\ $11144444 GPIOB_CRH !     \ PB15: PB14: PB13: Push-Pull
Output.
\ begin                     \ Start Begin - Until loop.
\ 1 15 lshift GPIOB_ODR !   \ RED, STOP ON, Set Pin15 ON.
\ ." STOP    "              \ Prints STOP on the Terminal.
\ 25000000 0 do loop        \ Delay Loop, XXX to 0.
\ 1 14 lshift GPIOB_ODR !   \ AMBER, Wait ON, Set Pin14 ON.
\ ." AMBER    "             \ Prints AMBER on the Terminal.
\ 4200000 0 do loop         \ Delay Loop, XXX to 0.
\ 1 13 lshift GPIOB_ODR !   \ GREEN, GO ON, Set Pin13 ON.
\ ." GO    "                \ Prints GO on the Terminal.
\ 25000000 0 do loop        \ Delay Loop, XXX to 0.
\ 1 14 lshift GPIOB_ODR !   \ AMBER, Wait ON, Set Pin14 ON.
\ ." AMBER    "             \ Prints AMBER on the Terminal.
\ 4200000 0 do loop         \ Delay Loop, XXX to 0.
\ key? until                \ Ends if any Key pressed, or
Loops forever.
\ ;
\
\ Remove the First AMBER Set in this instance to sequence
RED-to-GREEN, without AMBER.
\ TEST IT !
\
\ You can actually make the Red-Amber both briefly ON, before
Green, it involves addressing
\ the GPIOB_ODR directly and with two bits set and you also
adjust a couple of delay tmings.
\ If there's enough interest you may see a Part II.
\
\ And you can have this Traffic word run from Boot-Up, Power
ON, without a PC but it requires
\ a little more information so you don't get into trouble
first. The advanced lesson again.
\ You could try the Common Problems page here, (Search Index)
pay attention to the warning
\ or you may end up having to re-flash your Blue Pill with
Mecrisp Stellaris all over again.
\ For now this should get you started using FORTH on this
```

```
Traffic Light project.
\
\ Enjoy.
```

C and Forth Examples

See Keil C v/s Svd2forth, Configuring Peripherals

Charles (Chuck) Moore the inventor of Forth

Source: https://www.forth.com/resources/forth-programming-language/

Moore’s programming career began in the late 1950s at the Smithsonian Astrophysical Observatory with programs to compute ephemerides, orbital elements, satellite station positions, etc. [Moore, 1958], [Veis, 1960].

In 1961, Moore received his BA in Physics from MIT and entered graduate school at Stanford. He also took a part-time programming position at the Stanford Linear Accelerator (SLAC), writing code to optimize beam steering for the (then) pending two-mile electron accelerator, using an extension of some of his prior work with least-squares fitting. A key outgrowth of this work was a program called CURVE, coded in Algol (1964), a general-purpose non-linear differential-corrections data fitting program. To control this program, he used an enhanced version of his interpreter, extended to manage a push-down stack for parameter passing, variables (with the ability to explicitly fetch and store values), arithmetic and comparison operators, and the ability to define and interpret procedures.

In 1965, he moved to New York City to become a free-lance programmer. Working in Fortran, Algol, Jovial, PL/I and various assemblers, he continued to use his interpreter as much as possible, literally carrying around his card deck and recoding it as necessary. Minicomputers appeared in the late ’60s, and with them teletype terminals, for which Moore added operators to manage character input and output. One project involved writing a Fortran-Algol translator and file-editing utilities. This reinforced for him the value of spaces between words, which were not required in Fortran source.

Newly married and seeking a small town environment, Moore joined Mohasco Industries in Amsterdam, NY, in 1968. Here he developed computer graphics programs for an IBM 1130 minicomputer with a 2250 graphic display. This computer had a 16-bit CPU, 8k RAM, his first disk, keyboard, printer, card reader/punch (used as disk backup!), and Fortran compiler. He added a cross-assembler to his program to generate code for the 2250, as well as a primitive editor and source-management tools. This system could draw animated 3-D images, at a time when IBM’s software for that configuration drew only static 2-D images. For fun, he also wrote a version of Spacewar, an early video game, and converted his Algol Chess program into the new language, now (for the first time) called FORTH. He was impressed by how much simpler it became.

The name FORTH was intended to suggest software for the fourth (next) generation computers, which Moore saw as being characterized by distributed small computers. The operating system he used at the time restricted file names to five characters, so the “U” was discarded. FORTH was spelled in upper case until the late 1970s because of the prevalence of upper-case-only I/O devices. The usage “Forth” was generally adopted when lower case became widely available, because the word was not an acronym.

Moore found the Forth-based 1130 environment for programming the 2250 superior to the Fortran environment in which the 1130 software was developed, so he extended it into an 1130 compiler. This added looping commands, the concept of keeping source in 1024-byte blocks and tools for managing them, and most of the compiler features we recognize in Forth today.

Most important, there was now a dictionary. Procedures now had names, and the interpreter searched a linked list of names for a match. Names were compiled with a count and three characters, a practice learned from the compiler writers of Stanford and which prevailed in Forth until the 1980s. Within a dictionary entry was a “code field” containing the address of code to be executed for that routine. This was an indirect threaded code implementation (see Section 5.2) and was in use five years before Dewar’s paper on indirect threaded coded appeared in Communications of the ACM [Dewar 1975]. The use of indirect threaded code was an important innovation, since an indirect jump was the only overhead once a word had been found. Dictionary entries could consist either of pointers to other “high level” routines or of machine instructions.

Finally, in order to provide a simple mechanism for nesting routines, a second stack called the “return stack” was added. The benefit of having a stack reserved for return addresses was that the other stack could be used freely for parameter passing, without having to be “balanced” before and after calls. The first paper on Forth was written at Mohasco [Moore, 1970a]. In 1970 Mohasco assigned Moore to an ambitious project involving a new Univac 1108 handling a network of leased lines for an order-entry system. He ported Forth onto the 1108, and arranged for it to interface to COBOL modules that did the transaction processing. The 1108 Forth was coded in assembler. It buffered input and output messages and shared the CPU among tasks handling each line. It also interpreted the input and executed the appropriate COBOL modules. This version of Forth added mechanisms for defining and managing tasks, and also an efficient scheme for managing disk block buffers similar to schemes in use today. Unfortunately, an economic downturn led Mohasco to cancel the 1108 project before completion. Moore immediately gave notice, then wrote an angry poem and a book on Forth [Moore, 1970b] that was never published. It described how to develop Forth software and encouraged simplicity and innovation.

Chips

I love Integrated Circuits (chips) so this is a page about them.

This Months HOT MCU

STM32F7x7

- Arm® 32-bit Cortex® -M7 CPU
- Up to 2 Mbytes of Flash
- SRAM: 512 Kbytes,
- 16 Kbytes I/D cache, allowing 0-wait state execution from embedded Flash and external memories, up to 216 MHz
- 462 DMIPS/2.14 DMIPS/MHz (Dhrystone 2.1)
- DSP instructions

Download `STM32F777-technical-manual`

This is my HOT MCU choice for August, the STM32F7x7 MCU.

```
Features

· Memories

     Up to 2 Mbytes of Flash memory organized
        into two banks allowing read-while-write

     SRAM: 512 Kbytes (including 128 Kbytes
        of data TCM RAM for critical real-time data)
        + 16 Kbytes of instruction TCM RAM (for
        critical real-time routines) + 4 Kbytes of
        backup SRAM
```

  - Flexible external memory controller with up to 32-bit data bus: SRAM, PSRAM, SDRAM/LPSDR SDRAM, NOR/NAND memories

- Dual mode Quad-SPI

- Graphics
  - Chrom-ART AcceleratorTM (DMA2D), graphical hardware accelerator enabling enhanced graphical user interface

  - Hardware JPEG codec

  - LCD-TFT controller supporting up to XGA resolution

  - MIPI® DSI host controller supporting up to 720p 30 Hz resolution

- Clock, reset and supply management
  - 1.7 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - Dedicated USB power
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration

- Low-power
  - Sleep, Stop and Standby modes
  - VBAT supply for RTC, 32×32 bit backup registers + 4 Kbytes backup SRAM

- 3×12-bit, 2.4 MSPS ADC: up to 24 channels

- Digital filters for sigma delta modulator (DFSDM), 8 channels / 4 filters

- 2×12-bit D/A converters

- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support

- Up to 18 timers: up to thirteen 16-bit (1x low-power 16-bit timer available in Stop mode) and two 32-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input. All 15 timers running up to 216 MHz. 2x watchdogs, SysTick timer

- Debug mode
    - SWD & JTAG interfaces
    - Cortex®-M7 Trace MacrocellTM

- Up to 168 I/O ports with interrupt capability
    - Up to 164 fast I/Os up to 108 MHz
    - Up to 166 5 V-tolerant I/Os

- Up to 28 communication interfaces
    - Up to 4 I2C interfaces (SMBus/PMBus)
    - Up to 4 USARTs/4 UARTs (12.5 Mbit/s, ISO7816 interface, LIN, IrDA, modem control)
    - Up to 6 SPIs (up to 54 Mbit/s), 3 with muxed simplex I2S for audio
    - 2 x SAIs (serial audio interface)
    - 3 × CANs (2.0B Active) and 2x SDMMCs
    - SPDIFRX interface
    - HDMI-CEC
    - MDIO slave interface

- Advanced connectivity
    - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
    - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
    - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII

- 8- to 14-bit camera interface up to 54 Mbyte/s

- Cryptographic acceleration: hardware acceleration for AES 128, 192, 256, triple DES, HASH (MD5, SHA-1, SHA-2), and HMAC

- True random number generator

```
· CRC calculation unit

· RTC: subsecond accuracy, hardware calendar

· 96-bit unique ID
```

A Most Beautiful Chip, the STM32L162RD

These are some pics I took with my cheap USB microscope, so they are not that special, but hopefully you’ll get the idea.

STM32L162RD

This chip has been sitting on my bench for a few months, so I cleaned it as best I could with Metholated Spirits before taking the pics below.

The actual chip came from arrow.com where I purchased 10 units for $0.90 AUD each as they were ‘overstocked’.

The package is a WLCSP64 and is approximately 4.5 mm x 4.5 mm on each side and 0.38 mm thick with 64 solder blobs for pins (BGA).

The entire chip is silicon and the size of the actual DIE, there is no plastic or any other case material, and no copper pins.

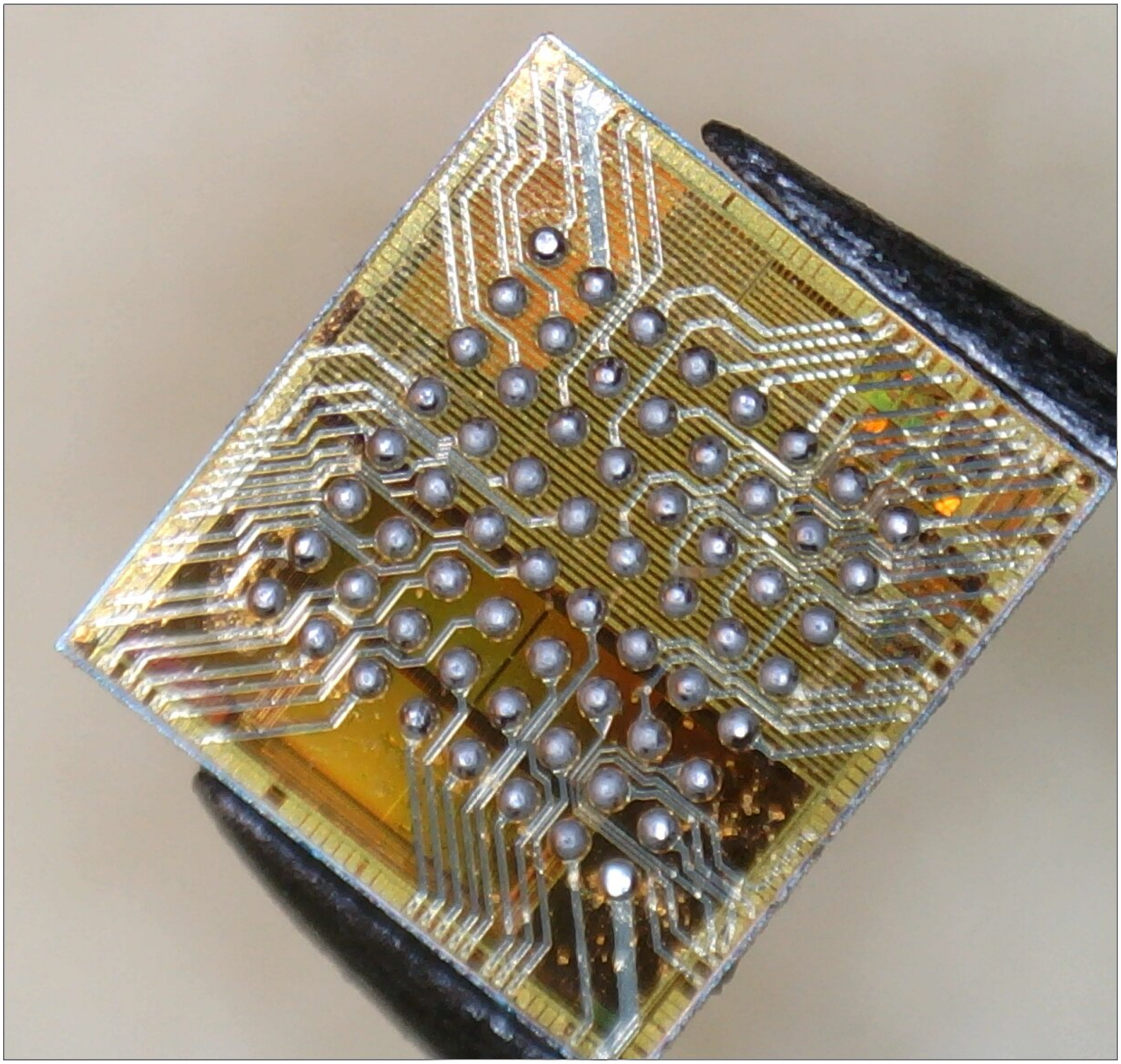
This to my mind represents the state of the art in miniaturization, where the connections are via solder blobs onto metallized tracks which connect to the pads on the die periphery.

STM32L162RD Bottom

When I looked at this chip under my 40x microscope it took my breath away because I could see under the silver and gold metallization into the chip itself. That’s right, the parallel lines which look a dull gold in my picture, looked shiny and like they are made of solid GOLD in the microscope

I’ve never seen anything as breathtaking as this before and I’m used to looking at chips with microscopes and lighting that make the “wire bonding” wires look the size of lamp posts.

4.5 x 4.5mm chip size, solder balls: 0.4mm apart



STM32L162RD Top

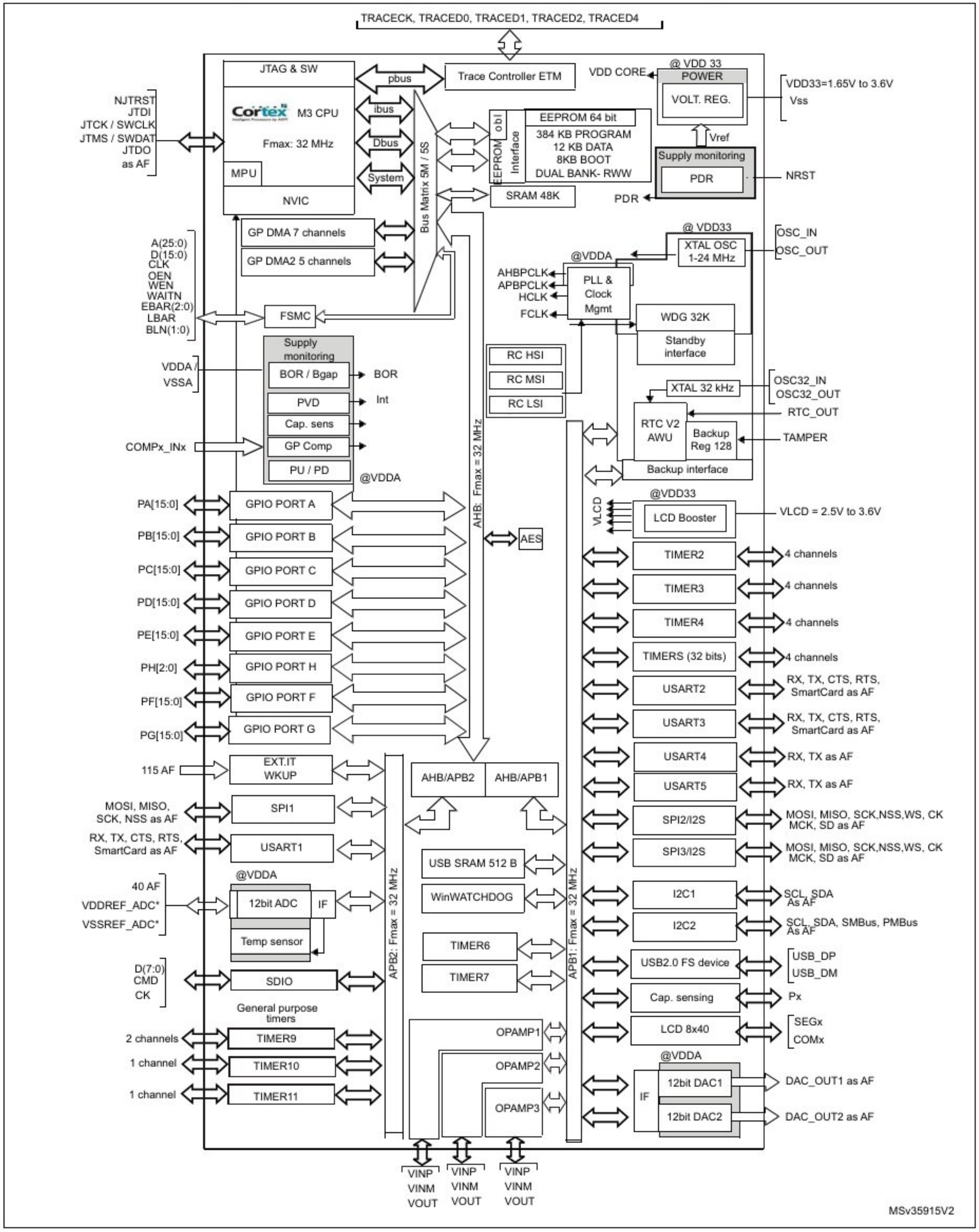
In this picture the laser/etch printing on the top of the chip can be seen. When it was newly taken out of the sealed shipping packaging, the top was like a mirror, flawless. Even after cleaning it’s still nothing like the unblemished new state.

4.5 x 4.5mm chip size

32L162RD
8N CG1 VG
SGP 427 V

Block Diagram

Figure 1. Ultra-low-power STM32L162xD block diagram


TRACECK, TRACED0, TRACED1, TRACED2, TRACED4
JTAG & SW
pbus
Trace Controller ETM
VDD CORE
@ VDD 33
POWER
VOLT. REG.
VDD33=1.65V to 3.6V
Vss
NJTRST
JTDI
JTCK / SWCLK
JTMS / SWDAT
JTDO
as AF
Cortex
M3 CPU
Fmax: 32 MHz
ibus
Dbus
System
MPU
NVIC
obl
EEPROM
Interface
EEPROM 64 bit
384 KB PROGRAM
12 KB DATA
8KB BOOT
DUAL BANK- RWW
Vref
Supply monitoring
PDR
NRST
PDR
SRAM 48K
Bus Matrix 5M / 5S
GP DMA 7 channels
GP DMA2 5 channels
A(25:0)
D(15:0)
CLK
OEN
WEN
WAITN
EBAR(2:0)
LBAR
BLN(1:0)
FSMC
@ VDD33
XTAL OSC
1-24 MHz
OSC_IN
OSC_OUT
@VDDA
AHBPCLK
APBPCLK
HCLK
FCLK
PLL &
Clock
Mgmt
WDG 32K
Standby
interface
Supply
monitoring
VDDA
VSSA
BOR / Bgap
BOR
PVD
Int
Cap. sens
GP Comp
PU / PD
COMPx_INx
RC HSI
RC MSI
RC LSI
XTAL 32 kHz
OSC32_IN
OSC32_OUT
RTC_OUT
RTC V2
AWU
Backup
Reg 128
TAMPER
Backup interface
@VDDA
AHB: Fmax = 32 MHz
@VDD33
VLCD
LCD Booster
VLCD = 2.5V to 3.6V
PA[15:0]
GPIO PORT A
PB[15:0]
GPIO PORT B
AES
PC[15:0]
GPIO PORT C
PD[15:0]
GPIO PORT D
PE[15:0]
GPIO PORT E
PH[2:0]
GPIO PORT H
PF[15:0]
GPIO PORT F
PG[15:0]
GPIO PORT G
TIMER2
4 channels
TIMER3
4 channels
TIMER4
4 channels
TIMERS (32 bits)
4 channels
USART2
RX, TX, CTS, RTS,
SmartCard as AF
USART3
RX, TX, CTS, RTS,
SmartCard as AF
USART4
RX, TX as AF
USART5
RX, TX as AF
115 AF
EXT.IT
WKUP
AHB/APB2
AHB/APB1
MOSI, MISO,
SCK, NSS as AF
SPI1
SPI2/I2S
MOSI, MISO, SCK,NSS,WS, CK
MCK, SD as AF
RX, TX, CTS, RTS,
SmartCard as AF
USART1
SPI3/I2S
MOSI, MISO, SCK,NSS,WS, CK
MCK, SD as AF
USB SRAM 512 B
@VDDA
40 AF
VDDREF_ADC*
VSSREF_ADC*
12bit ADC
IF
Temp sensor
WinWATCHDOG
I2C1
SCL, SDA
As AF
I2C2
SCL, SDA, SMBus, PMBus
As AF
APB2: Fmax = 32 MHz
APB1: Fmax = 32 MHz
TIMER6
TIMER7
USB2.0 FS device
USB_DP
USB_DM
D(7:0)
CMD
CK
SDIO
Cap. sensing
Px
General purpose
timers
2 channels
TIMER9
OPAMP1
LCD 8x40
SEGx
COMx
1 channel
TIMER10
OPAMP2
@VDDA
1 channel
TIMER11
IF
12bit DAC1
DAC_OUT1 as AF
OPAMP3
12bit DAC2
DAC_OUT2 as AF
VINP
VINM
VOUT
VINP
VINM
VOUT
VINP
VINM
VOUT
MSv35915V2

Capability

This chip is just JAM PACKED with peripherals, it is mind boggling!

- Ultra-low-power 32-bit MCU Arm® Cortex® -M3
- 384KB Flash
- 48KB SRAM
- 12KB EEPROM
- LCD, USB, ADC, memory I/F, AES
- Low Power, 305 nA Standby mode

```
Features                                           LQFP144
(20 × 20 mm)  UFBGA132     WLCSP64
                                                  LQFP100 (14
× 14 mm)  (7 × 7 mm)  (0.4 mm pitch)
· Ultra-low-power platform                         LQFP64 (10
× 10 mm)
    1.65 V to 3.6 V power supply
    -40°C to 105°C temperature range             · Memories
    305 nA Standby mode (3 wakeup pins)               384
Kbytes of Flash memory with ECC
    1.15 µA standby mode + RTC
(with 2 banks of 192 Kbytes enabling Rww
    0.475 µA Stop mode (16 wakeup lines)
capability)
    1.35 µA Stop mode + RTC                           48
Kbytes of RAM
    11 µA Low-power run mode                          12
Kbytes of true EEPROM with ECC
    230 µA/MHz Run mode                               128-
byte backup register
    10 nA ultra-low I/O leakage                       Memory
interface controller supporting
    8 µs wakeup time
SRAM, PSRAM and NOR Flash

· AES-128 bit encryption hardware accelerator      · LCD
driver for up to 8x40 segments
· Core: Arm® Cortex®-M3 32-bit CPU
Support contrast adjustment

Support blinking mode
    From 32 kHz up to 32 MHz max                      Step-up
converter on board
```

```
    1.25 DMIPS/MHz (Dhrystone 2.1)
    Memory protection unit                       · Rich
analog peripherals (down to 1.8V)
                                                       3x
operational amplifiers
· Reset and supply management                           12-
bit ADC 1 Msps up to 40 channels
    Low-power, ultrasafe BOR (brownout reset)         12-bit
DAC 2 ch with output buffers
       with 5 selectable thresholds                    2x
ultra-low-power comparators
    Ultra-low-power POR/PDR
(window mode and wakeup capability)
    Programmable voltage detector (PVD)
                                                  · DMA
controller 12x channels
· Clock sources
    1 to 24 MHz crystal oscillator               · 12x
peripheral communication interfaces
    32 kHz oscillator for RTC with calibration        1x USB
2.0 (internal 48 MHz PLL)
    High speed internal 16 MHz factory-               5x
USARTs
       trimmed RC (+/- 1%)                             Up to
8x SPIs (2x I2S, 3x 16 Mbit/s)
    Internal low power 37 kHz RC                      2x I2Cs
(SMBus/PMBus)
    Internal multispeed low power 65 kHz to           1x SDIO
interface
       4.2 MHz
    PLL for CPU clock and USB (48 MHz)           · 11x
timers: 1x 32-bit, 6x 16-bit with up to 4

IC/OC/PWM channels, 2x 16-bit basic timers,
· Pre-programmed bootloader                            2x
watchdog timers (independent and window)
    USB and USART supported
                                                  · Up to 34
capacitive sensing channels
· Development support
    Serial wire debug supported                  · CRC
calculation unit, 96-bit unique ID
    JTAG and trace supported
```

```
· Up to 116 fast I/Os (102 I/Os 5V tolerant), all
   mappable on 16 external interrupt vectors
```

The Perfect Chip Right?

So is it the perfect chip for embedded usage ?

Probably not, but it’s sure pretty, and perhaps represents the state of the art because I don’t think anything else could be packed into this tiny space.

This type of package is quite fragile, open to light so needs to be sealed underneath after reflow soldering. The solder balls are VERY close to the silver tracks (which may be non conductive on the surface as they cross over each other at a few places on the chip), so I imagine BEST BGA practices would be needed to use it.

I could easily scrape the metallization off the base with a knife using virtually no force tho this is not a fair criticism because I’m a 100 ton giant to this tiny silicon chip.

Runs Mecrisp-Stellaris?

Of course it does, or I wouldn’t have purchased it and made this page.

Cloning, Manual

See also

New version 5Jan2020

This page details cloning a Mecrisp-Stellaris image from a existing STM32F MCU or Board, complete with the Dictionary, Kernel, your user Words, the init Turnkey Word, … everything.

This reproduces exactly the same bootable system on another STM32F MCU or Board simply by flashing the cloned image to it.

1. Download the cloning file. It’s named “f051-clone.fs” as I tested it on a STM32F051 Discovery Board.

Download **`f051-clone.fs`**

Note

This code is essentialy code by Mattthias Koch found in his Mecrisp-Stellaris distribution in mecrisp-stellaris-x.x.x/lpc1114fn28/hexdump.txt

2. Set up something to capture the iHex file that will be generated by f051-clone.fs In the example I used the e4thcom terminal and the Unix ‘tee’ command to clone the terminal text into a logfile. You could use Picocom just as easily, or a logfile if your GUI serial terminal application has one.

```
e4thcom -t mecrisp -d ttyUSB0 -b B115200  |  tee e4thcom.log
```

3. Load f051-clone.fs onto your chip/board and run ‘clone’, you should see something like this:-

```
clone

:100000008C0300207548000005470000054700000EC
:10001000000000000000000000000000000000000E0
:1000200000000000000000000000000002F4700005A
... lots more of the same
:108DF000000000270720000B5FBF72EFAF9F758FC7C
:108E0000043E2E732000F9F7F8FCF9F7EAFB00BDE9
:108E1000FFFFFFFF60080570722E667370470000049
:00000001FF
```

4. Remove any log preamble and postamble then save it as clone.hex
5. Create a binary file from the iHex file

```
arm-none-eabi-objcopy -I ihex -O binary clone.hex clone.bin
```

6. Flash the binary to your STM32F chip/board

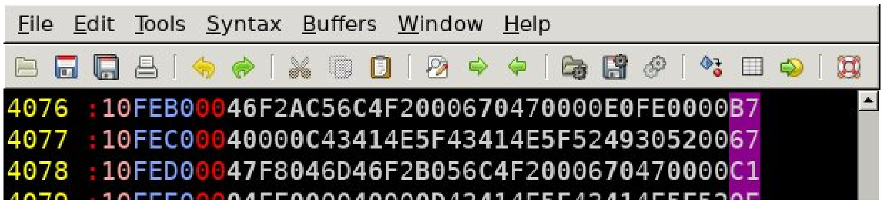
```
st-flash write clone.bin 0x08000000
```

7. Test the Clone
8. Optional; make a MD5SUM of your Clone binary, as it may come in handy later to find the exact same binary.

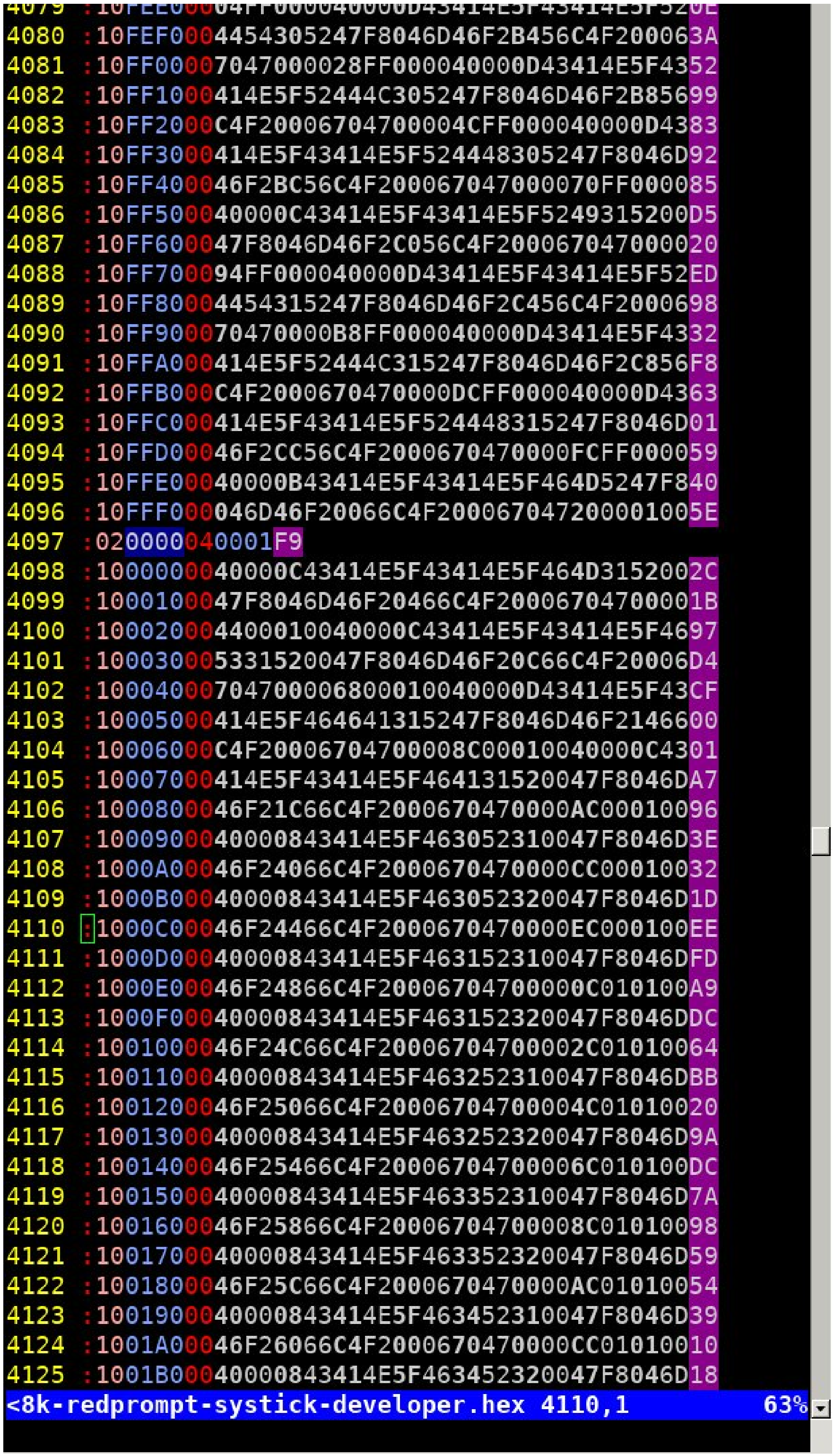
```
md5sum clone.bin > clone.bin.md5sum
```

Finished!

iHex



```
:10FEB00046F2AC56C4F2000670470000E0FE0000B7
:10FEC00040000C43414E5F43414E5F5249305200067
:10FED00047F8046D46F2B056C4F2000670470000C1
```



```
:10FEF0004454305247F8046D46F2B456C4F200063A
:10FF00007047000028FF000040000D43414E5F4352
:10FF1000414E5F52444C305247F8046D46F2B85699
:10FF2000C4F2000670470004CFF000040000D4383
:10FF3000414E5F43414E5F524448305247F8046D92
:10FF400046F2BC56C4F20006704700070FF000085
:10FF500040000C43414E5F43414E5F5249315200D5
:10FF600047F8046D46F2C056C4F2000670470000020
:10FF700094FF000040000D43414E5F43414E5F52ED
:10FF80004454315247F8046D46F2C456C4F2000698
:10FF900070470000B8FF000040000D43414E5F4332
:10FFA000414E5F52444C315247F8046D46F2C856F8
:10FFB000C4F20006704700000DCFF000040000D4363
:10FFC000414E5F43414E5F524448315247F8046D01
:10FFD00046F2CC56C4F200067047000FCFF000059
:10FFE00040000B43414E5F43414E5F464D5247F840
:10FFF000046D46F20066C4F20006704720000100005E
:020000040001F9
:1000000040000C43414E5F43414E5F464D3152002C
:1000100047F8046D46F20466C4F200067047000001B
:100020004400010040000C43414E5F43414E5F4697
:100030005331520047F8046D46F20C66C4F20006D4
:100040007047000068000100400000D43414E5F43CF
:10005000414E5F464641315247F8046D46F2146600
:10006000C4F2000670470008C00010040000C4301
:10007000414E5F43414E5F464131520047F8046DA7
:1000800046F21C66C4F200067047000AC00010096
:1000900040000843414E5F463052310047F8046D3E
:1000A00046F24066C4F200067047000CC00010032
:1000B00040000843414E5F463052320047F8046D1D
:1000C00046F24466C4F200067047000EC000100EE
:1000D00040000843414E5F463152310047F8046DFD
:1000E00046F24866C4F2000670470000C010100A9
:1000F00040000843414E5F463152320047F8046DDC
:1001000046F24C66C4F20006704700002C01010064
:1001100040000843414E5F463252310047F8046DBB
:1001200046F25066C4F20006704700004C01010020
:1001300040000843414E5F463252320047F8046D9A
:1001400046F25466C4F20006704700006C010100DC
:1001500040000843414E5F463352310047F8046D7A
:1001600046F25866C4F20006704700008C01010098
:1001700040000843414E5F463352320047F8046D59
:1001800046F25C66C4F2000670470000AC01010054
:1001900040000843414E5F463452310047F8046D39
:1001A00046F26066C4F2000670470000CC01010010
:1001B00040000843414E5F463452320047F8046D18
<8k-redprompt-systick-developer.hex 4110,1                63%
```

Cloning, Integrated

See also

New version 5Jan2020

In Cloning, manual I show how to clone a working Mecrisp-Stellaris MCU using 8 manual steps, which is easy enough, but required a Serial Terminal with logging and so on. Boring, slow, and error prone.

Cloning, A New Method

In this article I will show a much easier single step method.

There is also a YouTube Video at: https://www.youtube.com/watch?v=od4pkkT9sS8

Included is a cloned kernel image of my default development kernel for a STM32F051, used in the STM32F0 Discovery Board. It contains a number of development aids, see the kernel README.txt for the contents and notes.

Note

This, like all my Forth apps, is designed for Unix, developed on FreeBSD, and should run on Linux, OpenBsd etc. The usual default Unix utils are used and most should already be present on your Unix system.

Dependencies

- Gnu Screen
- DD
- arm-none-eabi-objcopy
- st-flash
- MD5 of some kind if you want to fingerprint the Kernel Image
- clone.sh
- f051-clone.fs

Operation

1. Install “clone.sh” in your path
2. Install “f051-clone.fs” in your path
3. Have a OPEN Interactive GnuScreen session to the development board/chip
4. Execute “clone.sh” … done.

Note

Clone will produce a binary named after the MD5sum, which is symlinked to ‘clone.bin’

Downloads

Download `clone.sh`

Download `f051-clone.fs`

Download `f0-disco-48mhz.fs`

Download Kernel `ea85d79924b9c367f125f4fa12a6c6e3.bin` 64KB

Download Wordlist `ea85d79924b9c367f125f4fa12a6c6e3.words.txt` 57KB

Download README

ea85d79924b9c367f125f4fa12a6c6e3.README.txt 3.5KB

clone.sh

```
#!/bin/sh
# Program Name: clone.fs
# Date: Sat 5 Jan 2019 03:01:34 AEDT
# Copyright 2018  t.porter <terry@tjporter.com.au>, licensed under the GPL
# For Mecrisp-Stellaris by Matthias Koch.
# https://sourceforge.net/projects/mecrisp/
# Dependencies
# ------------
# * Gnu Screen
# * DD
# * arm-none-eabi-objcopy
# * st-flash
# * MD5 of some kind if you want to fingerprint the Kernel Image
#
# Operation
# ---------
#
# 1) Install "clone.sh" in your path
# 2) Have a OPEN Interactive GnuScreen session to the development board/chip
#    xterm -title 'GnuScreen@cuaU0:460800 bps |  CTRL A :quit |  ' -e screen -t SCREEN1 -S SCREEN1  dev/cuaU0 460800,crtscts &
# 3) Execute "clone.sh" ... done.


AUTHOR="terry porter"  # <--- CHANGE ME TO YOUR NAME !!!
SCREENAME="SCREEN1"
PROJECT=`basename $PWD`
LIBRARY=/home/tp/projects/programming-languages/forth/mecrisp-stellaris/library
FILE_NAME_1=f051-clone.fs
LOGFILE="screenlog.0"
HEXFILE=$PROJECT.hex
BINARYFILE=$PROJECT.bin
LEADING_BYTES_TO_REMOVE=8
```

```
TRAILING_BYTES_TO_REMOVE=8
FINGERPRINT_APPLICATION="md5"
# ............................................. #

screen -p $SCREENAME -X stuff 'compiletoram\n'
screen -p $SCREENAME -X stuff 'reset\n'
screen -p $SCREENAME -X stuff 'f0-disco-48mhz\n'
screen -p $SCREENAME -X readreg x $LIBRARY/$FILE_NAME_1
screen -p $SCREENAME -X paste x
screen -p $SCREENAME -X logtstamp off
screen -p $SCREENAME -X log on
screen -p $SCREENAME -X stuff 'clone\n'
# sleep while logging the memory dump or screen will cut it
off while incomplete
sleep 10
screen -p $SCREENAME -X log off
BYTES=$(cat screenlog.0 | wc -c)
X=$((BYTES-LEADING_BYTES_TO_REMOVE))
Y=$((X-TRAILING_BYTES_TO_REMOVE))
dd if=$LOGFILE of=$HEXFILE bs=1 skip=$LEADING_BYTES_TO_REMOVE
count=$Y
arm-none-eabi-objcopy -I ihex -O binary $HEXFILE $BINARYFILE
$FINGERPRINT_APPLICATION $BINARYFILE > $BINARYFILE.md5sum
rm $LOGFILE $HEXFILE
md5name=$(md5 -q $BINARYFILE)
screen -p $SCREENAME -X log on
screen -p $SCREENAME -X stuff 'words\n'
sleep 10
screen -p $SCREENAME -X log off
mv screenlog.0 $md5name.words.txt
mv $BINARYFILE $md5name.bin
ln -s $md5name.bin clone.bin
touch $md5name.README.txt
echo $md5name.README: Specific Notes regarding Mecrisp-
Stellaris >$md5name.README.txt
echo Kernel $md5name.bin >>$md5name.README.txt
echo Created `date +%Y%m%d-%H%M` by $AUTHOR >>
$md5name.README.txt
echo --------------------------------------------------------
--------------------------- >> $md5name.README.txt
echo   >> $md5name.README.txt
echo All Finished!


# Flash the binary:
```

```
# st-flash erase
# st-flash write $BINARYFILE 0x08000000
```

Code Errors, Detecting

Forth is quite unique in that unlike other systems your source code is compiled, or interactively debugged on the target MCU.

With non Forth systems such as Arduino or Atollic TrueStudio, your source is compiled on a PC, code errors are announced by the compiler (GCC) you fix them, then Flash them to the MCU for testing. You may also have GDB for interactive debugging.

Note

Basic and some other operating systems such as Micropython also run on the MCU, but as far as I know, the source is interpreted and not compiled like Forth.

Whilst Forth sends error messages to you as they occur, a general purpose serial terminal won’t do anything about them and will keep uploading the source code long after the error has disappeared up the screen, and left you wondering why your shiny new code isn’t working …

Traditionally the upload speed was quite slow and the programmer watched for any errors.

To deal with this situation, some special purpose Forth Terminals have been created, which halt source code uploads when a error occurs. In my opinion the best of these is the e4thcom terminal by Manfred Mahlow, (other terminals of this kind are listed on the same page).

These special serial terminals for Forth also wait for the Forth end of line “OK” before sending the next line of code lest the MCU compiler choke up trying to compile code faster than it can. They also remove any comments so the MCU doesn’t have to waste time ignoring them and they offer #include statements to allow a ordered upload of source code files.

Sourcecode Errors

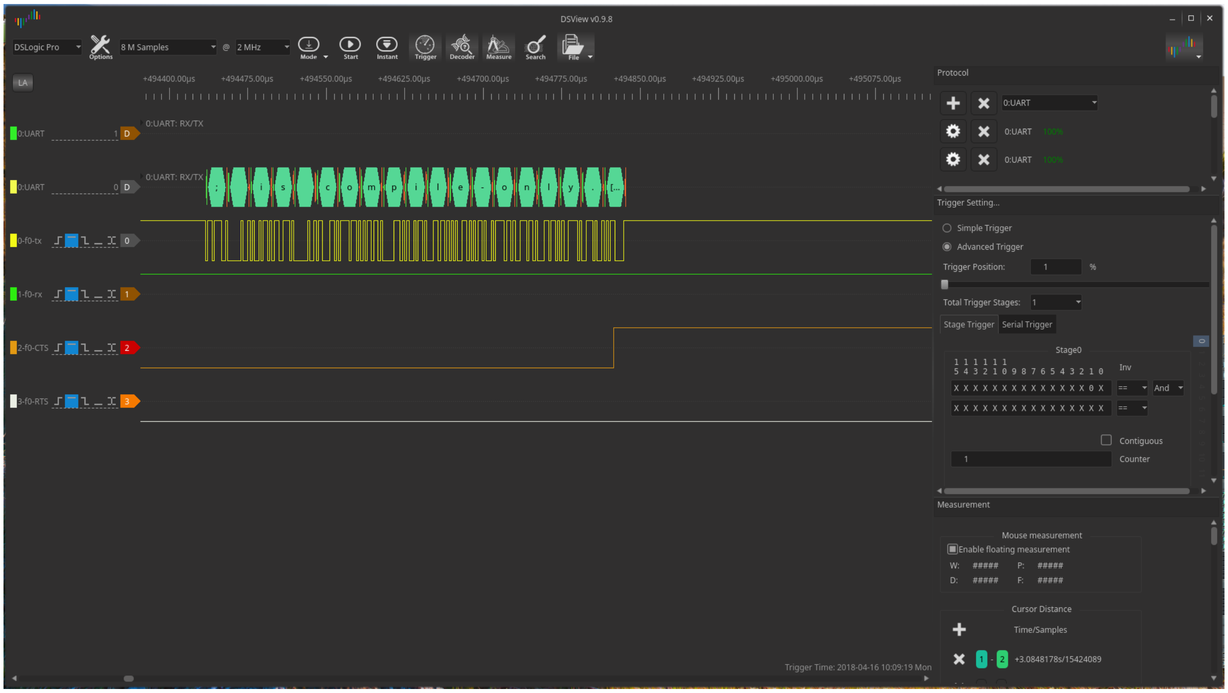
Detection and Action

I have been experimenting with a custom Mecrisp-Stellaris Kernel, running on a STM32F0 Discovery Board. When a Source Code Error (SCE) is detected by the Forth Compiler, it raises GPIOC-7, which in turn is connected to the STM32F0 Discovery Board’s CTS hardware handshaking pin.

This causes the STM32F0 Discovery Board serial communication to freeze, displaying the error at the user terminal so the programmer can say DOH!.

This system allows the use of general purpose serial terminals such as Picocom, Minicom, Gnu-Screen etc all running at higher speeds such as 460800 Baud.

In the picture below, a SCE has been detected. The SCE in this case is a “;” entered in the terminal manually and causing the error ” ; is compile only”. This in turn raises GPIOC-7, which is connected to the STM32F0 Discovery Board CTS pin, freezing the upload at this exact point.


DSView v0.9.8
DSLogic Pro
Options
8 M Samples
@ 2 MHz
Mode
Start
Instant
Trigger
Decoder
Measure
Search
File
LA
+494400.00µs
+494475.00µs
+494550.00µs
+494625.00µs
+494700.00µs
+494775.00µs
+494850.00µs
+494925.00µs
+495000.00µs
+495075.00µs
0:UART
0:UART: RX/TX
; i s c o m p i l e - o n l y . [...
0-f0-tx
1-f0-rx
2-f0-CTS
3-f0-RTS
Protocol
0:UART
100%
Trigger Setting...
○ Simple Trigger
● Advanced Trigger
Trigger Position: 1 %
Total Trigger Stages: 1
Stage Trigger
Serial Trigger
Stage0
Inv
And
☐ Contiguous
Counter
Measurement
Mouse measurement
☐ Enable floating measurement
W: ##### P: #####
D: ##### F: #####
Cursor Distance
Time/Samples
+3.0848178s/15424089
Trigger Time: 2018-04-16 10:09:19 Mon

Kernel Mod Features

This Kernel :-

1. Raises a GPIO pin on a source code error and turns on the STM32F0 Discovery Board BLUE LED.
2. Requires a board reset to clear the LED, but this is my usual development method because code is compiled into RAM and only moved to Flash when it is considered tested and stable. Ram is fast to reset, reboot and won't wear out.
3. Uses RTS hardware handshaking to halt the communications when a SCE occurs so it can be seen and fixed.
4. Communicates at 460800 Baud (8,N,1) to a USB 6 pin serial dongle with RTS and CTS on your pc. Because the comms faithfully halts on a SCE, you don't need to see your code uploading, so it may as well be as fast as possible.

5. Uses CTS hardware handshaking to the USB serial dongle to halt uploaded data while the MCU is compiling. This removes any requirement to add upload delaying *end of line delays* that are normally required with standard serial terminals such as Picocom.
6. Adds a carriage return after the Mecrisp-Stellaris line feed, so you don’t have to do that in your terminal as is normally required with Picocom .
7. Sets GPIOC 7,8 and 9 as OUTPUTS, so if you want to blink the LEDS connected to GPIOC 8 and 9, you don’t have to do this in your sourcecode.
8. The Kernel will run on any STM32F0 with at least 20K Flash, and uses USART1.

Note

Try it at a system clock speed of 48 MHz, I’m sure you will be impressed.

Running Picocom

Hardware handshaking is required and this example worked fine when I tested it.

```
picocom -f h -b 460800 /dev/<your usb-3.3v serial device>  --
omap delbs,crlf --send-cmd "ascii-xfr -sv"
```

F0 Disco Hardware Handshaking Connections

F0 Disco Signal	F0 Disco Port	USB Dongle

F0 Disco Signal	F0 Disco Port	USB Dongle
TX	PA10	RX
RX	PA9	TX
RTS	PA12	CTS

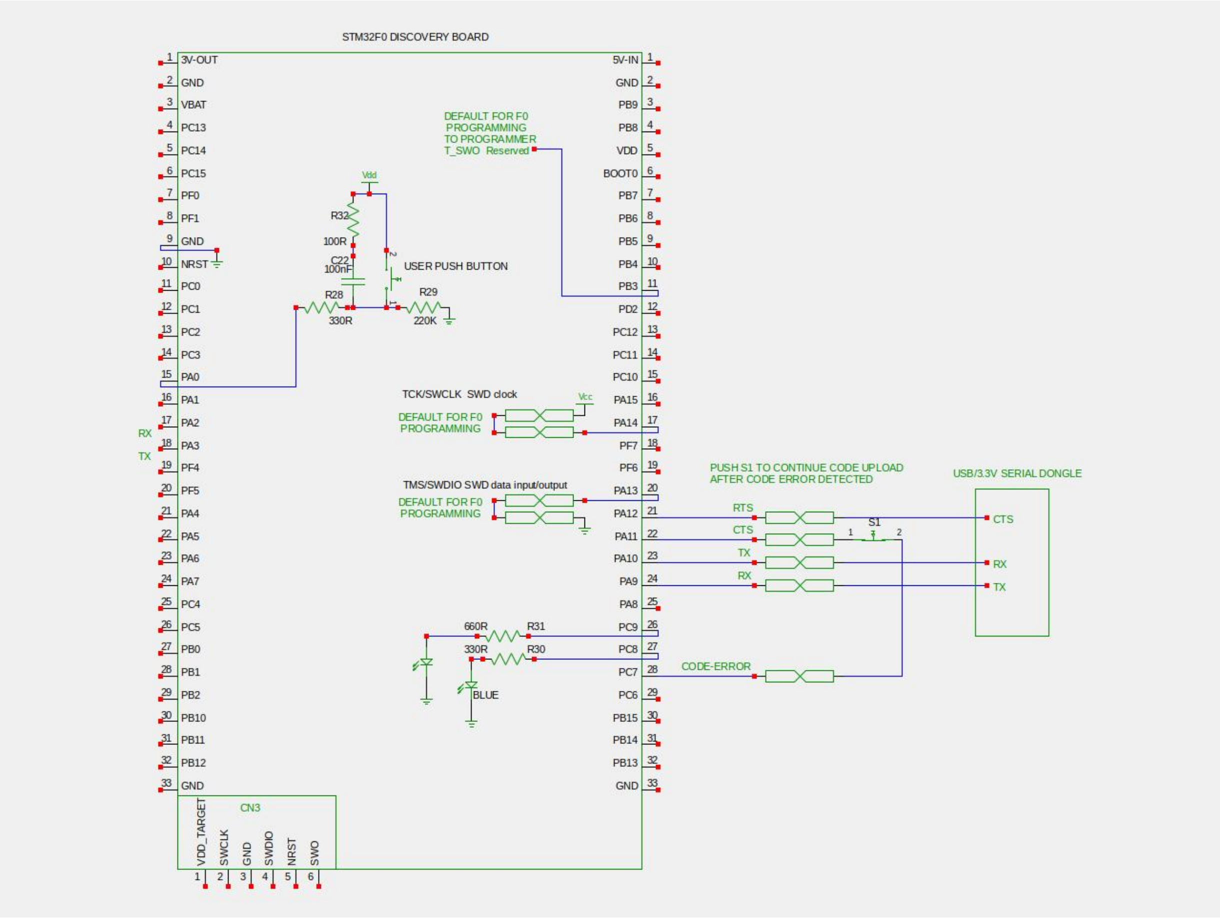
Note

See the schematic below for how a normally closed (NC) pushbutton switch is connected, so that communications is HALTED on a SCE, yet resumes when the button is pressed.

Schematic Diagram

This shows pin functions, and should be straightforward, please let me know if it’s not.

My STM32F0 Discovery Board ‘part’ shows on-board parts, such as the User Push Button circuitry inside it.


STM32F0 DISCOVERY BOARD
3V-OUT
GND
VBAT
PC13
PC14
PC15
PF0
PF1
GND
NRST
PC0
PC1
PC2
PC3
PA0
PA1
PA2
PA3
PF4
PF5
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PB10
PB11
PB12
GND
RX
TX
5V-IN
GND
PB9
PB8
VDD
BOOT0
PB7
PB6
PB5
PB4
PB3
PD2
PC12
PC11
PC10
PA15
PA14
PF7
PF6
PA13
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
GND
DEFAULT FOR F0 PROGRAMMING TO PROGRAMMER T_SWO Reserved
Vdd
R32
100R
C22
100nF
R28
330R
R29
220K
USER PUSH BUTTON
TCK/SWCLK SWD clock
Vcc
DEFAULT FOR F0 PROGRAMMING
TMS/SWDIO SWD data input/output
DEFAULT FOR F0 PROGRAMMING
PUSH S1 TO CONTINUE CODE UPLOAD AFTER CODE ERROR DETECTED
USB/3.3V SERIAL DONGLE
RTS
CTS
TX
RX
S1
CTS
RX
TX
660R
R31
330R
R30
BLUE
CODE-ERROR
CN3
VDD_TARGET
SWCLK
GND
SWDIO
NRST
SWO

Kernel

Grab it here, it’s only 20KB. Flash using your usual method.

Note

This kernel is a slightly modified mecrisp-stellaris-2.4.2-RA

DOWNLOAD: `discof0-rtscts-480600-nlcr-green-blue-pc7cerror.bin`

Code Reviews and Libraries

Code Review, is the act of collaborating with other programmers to check code for mistakes.

A library is a collection of code reused to develop software.

Disclaimer: I’m not a programmer, I’m a electronics technician/designer who stumbles and fumbles his way thru code. I enjoy the process of writing code but only when the end result is a working hardware solution.

Can Forth Code be Reviewed, are Libraries Useful ?

What follows are just a few of my thoughts on these matters, my opinion only, and I’m often wrong.

Every Forth Program is a Different Programming Language

Because Forth programs soon morph into the language of the problem they solve, technically every Forth program is a different programming language, unlike say Pascal or C where the language syntax is always the same from one program to the next.

Note

Would you ask a Haskell programmer to review your Perl code ?

Every Forth Implementation Targets a Specific MCU or MCU Family

Forth is as different as the hardware it runs on. Even in the same hardware family, say STM32xx, Mecrisp-Stellaris Forth may have different Kernel Words between models, depending on the hardware variations.

Examples

Set a BIT:-

Mecrisp-Stellaris for STM32Fxx

```
$48000800 constant PORTC_Base
PORTC_BASE $18 + constant PORTC_BSRR
4 constant Anode    \ PC4
1 Anode lshift PORTC_BSRR !   ( Anode high )
```

Flashforth for PIC:-

```
$ffc2 constant adcon0
%10 adcon0 mset \ Start conversion
```

Note

Would a Mecrisp-Stellaris user ask a Flashforth user to review their code ?

Most Forth Development Environments Are Different

Forth allows a programmer to design their own Development Environment.

Matthias Koch

Will write his memory mapped code something like this:-

```
$48000000 constant PORTA_Base
$48000400 constant PORTB_Base
$48000800 constant PORTC_Base
$48000C00 constant PORTD_Base
$48001400 constant PORTF_Base

PORTC_BASE $00 + constant PORTC_MODER    \ Reset 0 Port Mode
Register - 00=Input  01=Output  10=Alternate  11=Analog
PORTC_BASE $04 + constant PORTC_OTYPER   \ Reset 0 Port
Output type register - (0) Push/Pull vs. (1) Open Drain
PORTC_BASE $08 + constant PORTC_OSPEEDR  \ Reset 0 Output
Speed Register - 00=2 MHz  01=25 MHz  10=50 MHz  11=100 MHz
PORTC_BASE $0C + constant PORTC_PUPDR    \ Reset 0 Pullup /
Pulldown - 00=none  01=Pullup  10=Pulldown
PORTC_BASE $10 + constant PORTC_IDR      \ RO      Input Data
Register
PORTC_BASE $14 + constant PORTC_ODR      \ Reset 0 Output
Data Register
PORTC_BASE $18 + constant PORTC_BSRR     \ WO
PORTC_BASE $20 + constant PORTC_AFRL     \ Reset 0 Alternate
function  low register
PORTC_BASE $24 + constant PORTC_AFRH     \ Reset 0 Alternate
function high register
PORTC_BASE $28 + constant PORTC_BRR      \ Reset 0 Bit Reset
Register
```

He will then use this memory mapping code like so :-

```
4 constant Anode    \ PC4
5 constant Cathode  \ PC5

%01 Anode   2* lshift
%01 Cathode 2* lshift or PORTC_MODER bis! \ Set anode and
cathode as output
```

Note

> “The above code may seem simple enough, but try using this method to set PC4 and PC5 to mode INPUT ? Be sure to *test* your code and make sure it works first.

The Author

My memory mapping code is different, because I parse it automatically from CMSIS-SVD :-

```
$48000000 constant GPIOA_MODER ( GPIO port mode register )
$48000004 constant GPIOA_OTYPER ( GPIO port output type
register )
$48000008 constant GPIOA_OSPEEDR ( GPIO port output speed
register )
$4800000C constant GPIOA_PUPDR ( GPIO port pull-up/pull-down
register )
$48000010 constant GPIOA_IDR ( GPIO port input data register
)
$48000014 constant GPIOA_ODR ( GPIO port output data register
)
$48000018 constant GPIOA_BSRR ( GPIO port bit set/reset
register )
$4800001C constant GPIOA_LCKR ( GPIO port configuration lock
register )
$48000020 constant GPIOA_AFRL ( GPIO alternate function low
register )
$48000024 constant GPIOA_AFRH ( GPIO alternate function high
register )
$48000028 constant GPIOA_BRR ( Port bit reset register )
```

I also prefer to not have to consult the databook when manipulating Register Bitfields of more than ONE BIT, such as GPIOx_MODER which is TWO BITS wide, so I automatically generate Words to make it easy for me.

My thanks to Manfred Mahlow [manfredmahlow] for his bf! code.

```
: bf! ( bits bf-offs a-addr mask -- )
  swap >r over lshift >r lshift r@ and r> r@ bic! r> bis! ;
```

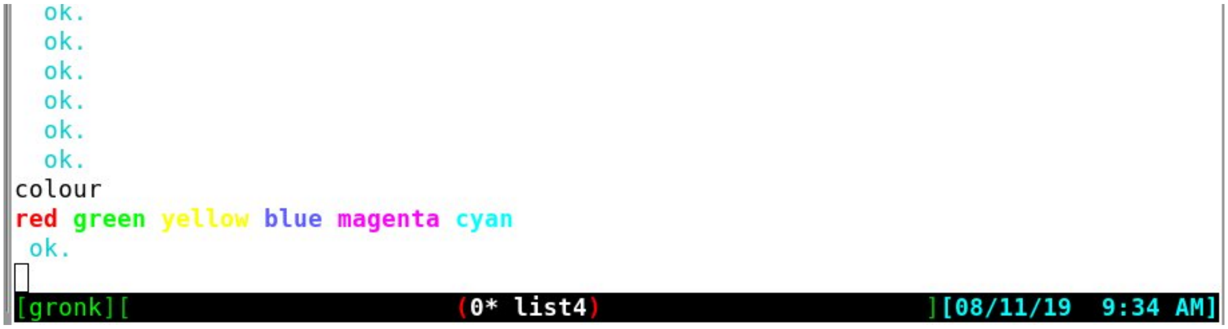
```
%00 constant INPUT
%01 constant OUTPUT
%10 constant ALT-FUNCT
%11 constant ANALOG

: GPIOA_MODER_MODER12 24 $48000000 %11 ;   ( PA12 mode )

OUTPUT GPIOA_MODER_MODER12 bf!     ( PA12 --> OUTOUT --> LED1 Anode )
1 GPIOA_ODR_ODR12 bf!              ( LED1 --> ON )
```

Note

Charles Moore, the inventor of forth said ” Don’t add extra code that your Forth program doesn’t need, add it later when you need it”.

Colour

Words to Provide Coloured Characters



```
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
colour
red green yellow blue magenta cyan
 ok.

[gronk][                    (0* list4)                    ][08/11/19  9:34 AM]
```

```
\ Program Name: colour-term.fs
\ Date: Aug 11 AEDT 2019
\ Copyright 2017  t.porter <terry@tjporter.com.au>, licensed under the GPL
\ For Mecrisp-Stellaris by Matthias Koch
\ This Program : Display colour characters on a serial terminal using ANSI escape codes
\
\ http://tldp.org/HOWTO/Bash-Prompt-HOWTO/x329.html
\ Black       0;30     Dark Gray     1;30
\ Blue        0;34     Light Blue    1;34
\ Green       0;32     Light Green   1;32
\ Cyan        0;36     Light Cyan    1;36
\ Red         0;31     Light Red     1;31
\ Purple      0;35     Light Purple  1;35
\ Brown       0;33     Yellow        1;33
\ Light Gray  0;37     White         1;37
\
\ Example for a terminal with a WHITE background:
\ : colour cr red ." red " green ." green " yellow ." yellow " blue ." blue " magenta ." magenta " cyan ." cyan " black cr ;
\ Example for a terminal with a BLACK background:
\ : colour cr red ." red " green ." green " yellow ." yellow " blue ." blue " magenta ." magenta " cyan ." cyan " white cr ;
```

```
compiletoflash

 : esc 27 emit ;
 : black    ( -- cursor colour )  esc ." [30;0m" ;
 : red      ( -- cursor colour )  esc ." [31;1m" ;
 : green    ( -- cursor colour )  esc ." [32;1m" ;
 : yellow   ( -- cursor colour )  esc ." [33;1m" ;
 : blue     ( -- cursor colour )  esc ." [34;1m" ;
 : magenta  ( -- cursor colour )  esc ." [35;1m" ;
 : cyan     ( -- cursor colour )  esc ." [36;1m" ;
 : white    ( -- cursor colour )  esc ." [37;1m" ;

compiletoram
```

Common Programs

Describes programs supplied with every Mecrisp-Stellaris Release. These are located in “mecrisp-stellaris-X.X.X/common”.

This page is a work in progress ….

All the programs below are loaded the usual way via a serial terminal

assembler-m0.txt

bitlog.txt

blocks.txt

calltrace.txt

charcomma.txt

conditional.txt

disassembler-m0.txt

Disassembles Cortex M0 Words into Assembly Language. Useful for optimising your Words etc.

Calling Word(s)

- see

```
: three-plus 3 + ;  ok.

3 three-plus . 6  ok.

see three-plus
20000342: B500  push { lr }
20000344: 3603  adds r6 #3
20000346: BD00  pop { pc }
 ok.
```

disassembler-m3.txt

Same as disassembler-m0.txt but for the Cortex M3

dump.txt

Reads (dumps) bytes or regions of memory.

Calling Word(s)

- dump16 (address – dump 16 bytes of memory)
- dump (address range – dump memory)

```
0 dump16
00000000 :  30 03 00 20 49 37 00 00   D9 35 00 00 D9 35 00 00
|  0.. I7..  .5...5.. |


0 $30 dump
00000000 :  30 03 00 20 49 37 00 00   D9 35 00 00 D9 35 00 00
|  0.. I7..  .5...5.. |
00000010 :  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
|  ........  ........ |
00000020 :  00 00 00 00 00 00 00 00   00 00 00 00 03 36 00 00
|  ........  .....6.. |
00000030 :  00 00 00 00 00 00 00 00   03 36 00 00 B7 35 00 00
|  ........  .6...5.. |

$30 $50 dump
00000030 :  00 00 00 00 00 00 00 00   03 36 00 00 B7 35 00 00
|  ........  .6...5.. |
00000040 :  03 36 00 00 03 36 00 00   03 36 00 00 03 36 00 00
|  .6...6..  .6...6.. |
00000050 :  03 36 00 00 27 36 00 00   4B 36 00 00 71 36 00 00
|  .6..'6..  K6..q6.. |
00000060 :  03 36 00 00 03 36 00 00   03 36 00 00 03 36 00 00
|  .6...6..  .6...6.. |
00000070 :  93 36 00 00 B7 36 00 00   DB 36 00 00 FD 36 00 00
|  .6...6..  .6...6.. |
00000080 :  21 37 00 00 03 36 00 00   03 36 00 00 03 36 00 00
|  !7...6..  .6...6.. |
```

editor.txt

examples.txt

experimental/

floored-divide.txt

fonts/

Flash

Error Message

Wrong address or data for writing flash !

Originates from ?

flash.s in your MCU source directory

Causes

1. Are you trying to write into the flash dictionary or outside the Forth core ? The Forth core address range is

set here
2. Is the address even ? it must be divisible by 2
3. Does the data at the Flash memory address contain -1, ie all $F ? ($FFFFFFFF is the erased state of the Flash on Cortex-M0)

graphics-unicode-8x16.txt

graphics-unicode-8x8.txt

graphics.txt

longcomment.txt

mecrisp-registergenerator/

multitask.txt

nvic.txt

profiler.txt

prompt.txt

pseudorandom.txt

quotations.txt

registergenerator/

romans.txt

sine.txt

sqrt.txt

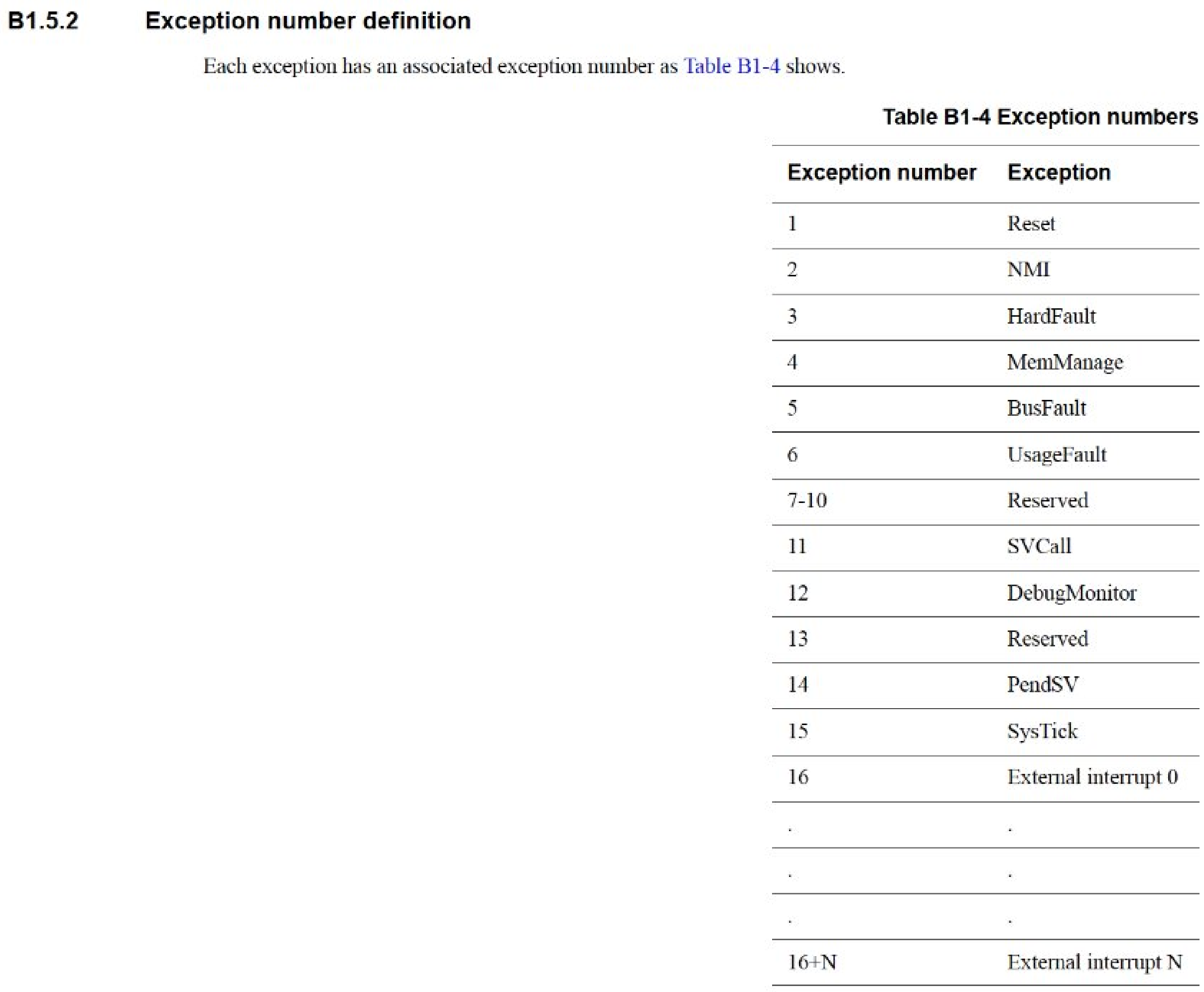
svd2forth-v2/

Exceptions

Note

This EXCEPTION article was written using a ST32F103C8 MCU (Cortex-M3) for the examples and testing.

- An EXCEPTION is defined in the ARM specification as “a condition that changes the normal flow of control in a program”
- An EXCEPTION is similar to a INTERRUPT.
- Cortex-M has many EXCEPTIONS and they generally occur when something bad happened, such as reading a memory location that doesn’t exist.
- The registers that deal with EXCEPTIONS reside in the ARM “Core Peripherals” System control block (SCB) peripheral and common across all Cortex-M models.
- SCB_ICSR @ $E000ED04 the “Interrupt control and state register” is relevant here.
- For STM32F103 see ST Micro Document PM0056, page 129/156



B1.5.2 Exception number definition

Each exception has an associated exception number as Table B1-4 shows.

Table B1-4 Exception numbers

Exception number	Exception
1	Reset
2	NMI
3	HardFault
4	MemManage
5	BusFault
6	UsageFault
7-10	Reserved
11	SVCall
12	DebugMonitor
13	Reserved
14	PendSV
15	SysTick
16	External interrupt 0
.	.
.	.
.	.
16+N	External interrupt N

Debugging

The Word “Calltrace.txt” attempts to list the cause of the EXCEPTION and also prevents the dreaded “Unhandled Interrupt 00000003 !” message printing endlessly until the MCU is reset.

Requires

mecrisp-stellaris-x.x.x/common/calltrace.txt from your Mecrisp-Stellaris release tarball.

Warning

calltrace.txt searches for the closest dictionary entry points to the addresses on the return stack and may give random results for values that aren’t return addresses. It is assumed that the programmer can decide from context which ones are correct.

Implementation

The “irq-fault” INTERRUPT must be configured to call the Word “ct-irq” in your program. When a EXCEPTION occurs, this attempts to show the cause.

```
' ct-irq irq-fault !
```

Example; Terminal Entry

This EXCEPTION was contrived by attempting to read non-existent memory at the Terminal. This is a very common occurrence during development.

```
$100000 @ .
```

Result

```
Unhandled Interrupt 00000003 !

[1 ] 0000006E 00100000
Calltrace:
00000000 0000C0DD  calltrace-handler
00000001 FFFFFFF9
00000002 00001824  @
00000003 00000220  --- MS RA 2.5.1 tpmod ---
00000004 00001824  @
00000005 00000020
00000006 FEFBAFF7
00000007 00004683  interpret
00000008 00001824  @
```

```
00000009 01000000  init.calltrace
0000000A 0000BF2D  redstack.prompt
0000000B 000046DD  quit
```

Note

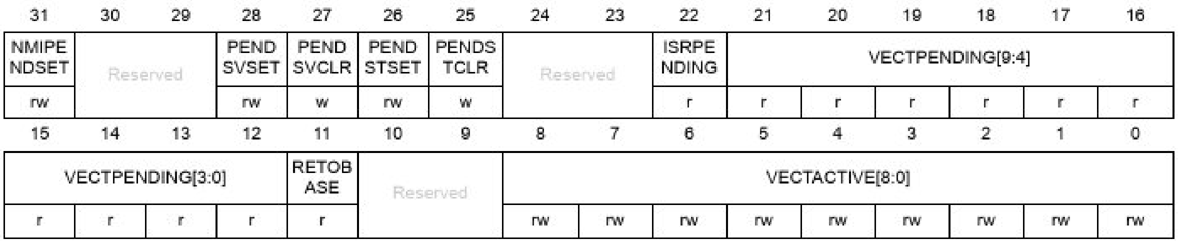
The MCU is HALTED at this point and will require a RESET as most EXCEPTIONS are not recoverable.

Description

1. The “HardFault” EXCEPTION #3 was raised
2. The stack contents are dumped, note the “00100000” which I had entered before the EXCEPTION was raised.
3. Calltrace was run (from the previously loaded “calltrace.txt”)
4. Some additional irrelevant information was also printed as per the warning above.

Customisation

A printout of the SCB_ICSR (at $E000ED04) register has been added to calltrace.txt for additional information.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NMIPENDSET	Reserved		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	Reserved		ISRPENDING	VECTPENDING[9:4]					
rw			rw	w	rw	w			r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VECTPENDING[3:0]				RETOBASE	Reserved		VECTACTIVE[8:0]								
r	r	r	r	r			rw	rw	rw	rw	rw	rw	rw	rw	rw

```
1000000 @ .

Unhandled Interrupt 00000003 !
```

```
[1 ] 0000006E 000F4240
Calltrace:
SCB_ICSR:$0400F803
3322222222221111111111
10987654321098765432109876543210
00000100000000001111100000000011

00000000 0000A73D  calltrace-handler
00000001 FFFFFFF9
00000002 00001824  @
00000003 00000220  --- MS RA 2.5.1 tpmod ---
00000004 00001824  @
00000005 00000020
00000006 FEFBAFF7
00000007 00004683  interpret
00000008 00001824  @
00000009 01000000  init.calltrace
0000000A 0000A56D  redstack.prompt
0000000B 000046DD  quit
```

Example; Program Execution

In this example the Word “cause-exception” is executed.

```
: cause-exception ( -- )
 1000000 @ .
;
```

Code

```
getid
cause-exception
getid-STM32F103CB-data
getid-GD32F103-data
xml
```

Result

```
Unhandled Interrupt 00000003 !

[0 ] 00000000
```

```
Calltrace:
SCB_ICSR:$0400F803
3322222222221111111111
10987654321098765432109876543210
00000100000000001111100000000011

00000000 0000A73B  calltrace-handler
00000001 FFFFFFF9
00000002 000F4200  init.calltrace
00000003 00000000
00000004 20000DDE  cause-exception
00000005 00000000
00000006 FEFBAFF7
00000007 00004683  interpret
00000008 20000DE6  cause-exception
00000009 01000000  init.calltrace
0000000A 0000A56D  redstack.prompt
0000000B 000046DD  quit
```

Description

1. The “HardFault” EXCEPTION #3 was raised
2. The stack contents are dumped, note the listing of the cause “cause-exception” after “calltrace-handler” is triggered making the reason for the EXCEPTION easy to find.
3. Calltrace was run (from the previously loaded “calltrace.txt”)
4. Some additional irrelevant information was also printed as per the warning above.

See also

Bugs or Benefits?

Contributing Code

If you would like to contribute your working Mecrisp-Stellaris code to this site for others to read and use, just email it to me at: <terry@tjporter.com.au> and I’ll add it to the “User Submitted Code” collection here.

There are 5 requirements:

1. Add the following HEADER FILE to the start of each submitted code file and edit it to suit.

DOWNLOAD: HEADER FILE

```
\ Program Name:  <program.txt> or <program.fs>
\ Date:
\ Copyright by: <your name>
\ License: <must be Open Source of some kind>
\ For Mecrisp-Stellaris by Matthias Koch.
https://sourceforge.net/projects/mecrisp/
\ Chip:
\ Board:
\ Clock: 8 MHz using the internal RC clock, unless otherwise
stated
\ Note: GPIO's A,B,C,D,E and UART1 are pre-enabled by the
Mecrisp-Stellaris Core.
\ All register names are CMSIS-SVD compliant: YES or NO ?
\ Program Description:
\
\
\ Requires these files be previously loaded:
\
\ -----------------------------------------------------------
-------------------------------
```

2. Your file needs to be actual Mecrisp-Stellaris code in text, the same as you upload to your MCU, no markup language, no HTML etc.

3. Try and make sure all dependencies are met/included for your code to run.
4. Everything must be in English please.
5. If you update anything, fix a bug in the file at a later date just email me a complete file as the replacement with the same file name etc as I don’t edit submitted code.

Contributors Go Here

- Terry Porter <terry@tjporter.com.au> and < tp on irc.freenode: ##tech40+ >: Maintainer of this site, author of svd2forth-v2
- Krzysztof Foltman: Author of Forth Explorer written just for Mecrisp Stellaris.
- Qih for his Morse Code program
- McKinnon Ryall vatrat.dev@gmail.com for cleaning up the stm32loader.py bootloader
- Ivan Dimitrov for proof reading the Dictionary listing.

Critical Systems

Toyota

Two articles about the very interesting case of Toyota and a perceived lack of critical safety engineering in the 2005 Toyota Camry.

They serve as a warning to all embedded designers that safety has to be number one in our list of priorities.

Safety Research and Strategies Inc

Posted on Thursday, Nov 7th, 2013

Source: http://www.safetyresearch.net/blog/articles/toyota-unintended-acceleration-and-big-bowl-%E2%80%9Cspaghetti%E2%80%9D-code

Last month, Toyota hastily settled an Unintended Acceleration lawsuit – hours after an Oklahoma jury determined that the automaker acted with “reckless disregard,” and delivered a $3 million verdict to the plaintiffs – but before the jury could determine punitive damages.

What did the jury hear that constituted such a gross neglect of Toyota’s due care obligations? The testimony of two plaintiff’s experts in software design and the design process gives some eye-popping clues. After reviewing Toyota’s software engineering process and the source code for the 2005 Toyota Camry, both concluded that the system was defective and dangerous, riddled with bugs and gaps in its failsafes that led to the root cause of the crash.

Bookout and Schwarz v. Toyota emanated from a September 2007 UA event that caused a fatal crash. Jean Bookout and her friend and passenger Barbara Schwarz were exiting Interstate Highway 69 in Oklahoma, when she lost throttle control of her 2005 Camry. When the service brakes would not stop her speeding sedan, she threw the parking brake, leaving a 150-foot skid mark from right rear tire, and a 25-foot skid mark from the left. The Camry, however, continued speeding down the ramp and across the road at the bottom, crashing into an embankment. Schwarz died of her injuries; Bookout spent five months recovering from head and back injuries.

Attorney Graham Esdale, of Beasley Allen, who represented the plaintiffs is the first to say that the Bookout verdict – in some measure – rested on those two black skid marks scoring the off- ramp.

“Toyota just couldn’t explain those away,” Esdale said. “The skid marks showed that she was braking.”

The jury was very attentive, despite the technical discussions that dominated the testimony. After the jury learned that the case had been settled, jurors asked Judge Patricia Parrish if they could stay and discuss the trial. A dozen jurors, Judge Parrish, and the plaintiff’s lawyers discussed it. Esdale says that it was obvious from that conversation that the jury was poised to punish Toyota for its conduct and cover-up.

Skid marks notwithstanding, two of the plaintiffs’ software experts, Phillip Koopman, and Michael Barr, provided fascinating insights into the myriad problems with Toyota’s software development process and its source code – possible bit flips, task deaths that would disable the failsafes, memory corruption, single-point failures, inadequate protections against stack overflow and buffer overflow, single-fault containment regions, thousands of global variables. The list of deficiencies in process and product was lengthy.

Michael Barr, a well-respected embedded software specialist, spent more than 20 months reviewing Toyota’s source code at one of five cubicles in a hotel-sized room, supervised by security guards, who ensured that entrants brought no paper in or out, and wore no belts or watches. Barr testified about the specifics of Toyota’s source code, based on his 800-page report. Phillip Koopman, a Carnegie Mellon University professor in computer engineering, a safety critical embedded systems specialist, authored a textbook, Better Embedded System Software, and performs private industry embedded software design reviews – including in the automotive industry – testified about Toyota’s engineering safety process. Both used a programmer’s derisive term for what they saw: spaghetti code – badly written and badly structured source code.

Barr testified:

There are a large number of functions that are overly complex. By the standard industry metrics some of them are untestable, meaning that it is so complicated a recipe that there is no way to develop a reliable test suite or test methodology to test all the possible things that can happen in it. Some of them are even so complex that they are what is called unmaintainable, which means that if you go in to fix a bug or to make a change, you’re likely to create a new bug in the process. Just because your car has the latest version of the firmware – that is what we call embedded software – doesn’t mean it is safer necessarily than the older one….And that conclusion is that the failsafes are inadequate. The failsafes that they have contain defects or gaps. But on the whole, the safety architecture is a house of cards. It is possible for a large percentage of the failsafes to be disabled at the same time that the throttle control is lost.

Even a Toyota programmer described the engine control application as “spaghetti-like” in an October 2007 document Barr read into his testimony.

Koopman was highly critical of Toyota’s computer engineering process. The accepted, albeit voluntary, industry coding standards were first set by Motor Industry Software Reliability Association (MISRA) in 1995. Accompanying these rules is an industry metric, which equates broken rules with the introduction of a number of software bugs: For every 30 rule violations, you can expect on average three minor bugs and one major bug. Toyota made a critical mistake in declining to follow those standards, he said.

When NASA software engineers evaluated parts of Toyota’s source code during their NHTSA contracted review in 2010, they checked 35 of the MISRA-C rules against the parts of the Toyota source to which they had access and found 7,134 violations. Barr checked the source code against MISRA’s 2004 edition and found 81,514 violations.

Toyota substituted its own process, which had little overlap with the industry standard. Even so, Toyota’s programmers often broke their own rules. And they failed to keep adequate track of their departures from those rules – and the justification for doing so, which is also standard practice. Koopman testified that if safety is not baked into the recipe in the process of creating the product, it cannot be added later.

“You have to exercise great care when you’re doing safety critical software. You can’t just wing it. And Toyota exercised some care, but they did not reach the level of accepted practice in how you need to design safety critical systems,” he said.

One of the biggest safety standards Toyota broke was allowing single point failures within its system. (Single point failure refers to a piece of hardware or software that has complete control over whether a system is safe or not—such as a single-engine airplane.) Koopman testified:

“If there is a single point of failure, by every safety standard I have ever seen, it is by definition unsafe, and no amount of countermeasures, no amount of failsafes will fix that. They will reduce how often it happens, but it won’t completely fix it. Because we have millions of vehicles out there, it will find a way to fail that you didn’t think of, and it will fail.”

Other egregious deviations from standard practice were the number of global variables in the system. (A variable is a location in memory that has a number in it. A global variable is any piece of software anywhere in the system can get to that number and read it or write it.) The academic standard is zero. Toyota had more than 10,000 global variables.

“And in practice, five, ten, okay, fine. 10,000, no, we’re done. It is not safe, and I don’t need to see all 10,000 global variables to know that that is a problem,” Koopman testified.

Other important design process errors Barr and Koopman identified were an absence of a peer code review, and Toyota’s failure to check the source code of its second CPU, supplied by Denso —even as executives assured Congress and NHTSA that the cause of UA couldn’t be in the engine software.

Barr testified to some of the vehicle behavior malfunctions caused by the death of tasks within the CPU, and concluded that Bookout’s UA was more likely than not caused by the death of a redacted-name task, called Task X at trial. Barr dubbed it “the kitchen-sink” task, because it controlled a lot of the vehicle’s functions, including throttle control; the cruise control – turning it on, maintain the speed and turning it off – and many of the failsafes on the main CPU.

He was critical of Toyota watchdog supervisor – software to detect the death of a task – design. He testified that Toyota’s watchdog supervisor “is incapable of ever detecting the death of a major task. That’s its whole job. It doesn’t do it. It’s not designed to do it.”

Instead, Toyota designed it to monitor CPU overload, and, Barr testified: “it doesn’t even do that right. CPU overload is when there’s too much work in a burst, a period of time to do all the tasks. If that happens for too long, the car can become dangerous because tasks not getting to use the CPU is like temporarily tasks dying.”

Barr also testified that Toyota’s software threw away error codes from the operating system, ignoring codes identifying a problem with a task. At trial, Barr said:

And task death, although I focused a lot of task X here, because it does so much and it does throttle control and it does failsafe, it’s pretty important, but there is [redacted] tasks and they can die in different combinations. It could be task 3 and task X, or task 3and task 7 and task X, or just task 9. And those can cause an unpredictable range of vehicle misbehaviors. It turns out that unintended acceleration is just the most dangerous thing your car can do when it malfunctions.

Even if you were to dismiss their conclusions as nothing but paid-for expert testimony, Koopman and Barr's assessment about software errors as a possible UA root cause go a long way in explaining so much: how Toyota's system could fail and leave no trace; why we are still seeing UAs in late model Toyota vehicles and why Toyota can't seem to fix it with floor mat and pedal recalls; how it could get away with hiding some of the root causes of UA events for so long.

Their descriptions of the incredible complexity of Toyota's software also explain why NHTSA has reacted the way it has and why NASA never found a flaw it could connect to a Toyota's engine going to a wide open throttle, ignoring the driver's commands to stop and not set a diagnostic trouble code. For one, Barr testified, the NASA engineers were time limited, and did not have access to all of the source code. They relied on Toyota's representations – and in some cases, Toyota misled NASA. For example, NASA was under the false belief that Toyota had designed in hardware bit flip protections called Error Detection and Correction Codes, (EDAC). The 2005 Camry for example did not have EDAC, Barr testified, but in an email Toyota told NASA that it did. At trial he said:

NASA didn't know that that wasn't there. It wasn't there in the 2005 Camry. And so if the bit-flip occurred, there would be no hardware mechanism to find it. And if it occurred in a critical value that was not mirrored, there would be no software protections against it. So the conclusion here is that there are critical variables in which bits could flip.

Their testimony explains why it would be near impossible for NHTSA to ever pin an electronic failure on a problem buried in software. NHTSA didn’t even have any software engineers on ODI’s staff during the myriad Toyota UA investigations. They have no real expertise on the complexities that actually underpin all of the safety-critical vehicle functions of today’s cars. It’s as if ODI engineers are investigating with an abacus, a chisel and a stone tablet. One begins to understand the agency’s stubborn doubling, tripling, quadrupaling down on floor mats and old ladies as explanations for UA events.

But even if NHTSA did have this expertise, the software piece is so complex ODI would never have the time or budget to assess an automaker’s source code. This is why we keep harping on the need for NHTSA to write a functional safety regulation – under its own steam or Congressional mandate.

We are posting preliminary drafts of Koopman’s (part 1 and part 2) and Barr’s trial testimony, along with Barr’s slides – long, but well worth a read for anyone interested in understanding more about embedded software systems in automobiles and how not to design one; where NHTSA went wrong: and the unbelievably shaky software at the foundation of Toyota’s electronic architecture.

Normally, one associates a company’s desire to shield trade secrets with the protection of something valuable. That something, one presumes, is the technology itself – the secret recipe a company uses in making its product. Rather than protecting the automotive equivalent of formula for Coke, the testimony of Koopman and Barr suggest that Toyota really wanted to hide was its formula for disaster. Consider the contents of a September 2007 email among Toyota employees:

“‘In truth technology such as failsafe is not part of the Toyota’s engineering division’s DNA,’ ” Barr read in court. “And it continues, ‘But isn’t it good that it is recognized as one of the major strengths of Toyota and its system controls industry.’ And then I highlighted also the portion that says, ‘Continuing on as is would not be a good thing.’”

BOOKOUT V. TOYOTA

2005 Camry L4 Software Analysis

`download slides pdf`

Data Stack

A Succint Stack Description by Dave Wyland

The data stack is the unusual part of Forth. It’s primary function is to pass operands to and from the Forth words. You use the stack for the parameters rather than enumerating them explicitly in the call structure as is required in almost all languages from Fortran to C and onward.

It turns out that the stack is not a bug, it is a feature. It has benefits:

1. You do not have to name your parameters to pass them. for example, a parameter on the stack can be (and often is) the un-named result of a previous calculation.
2. You do not have to name your parameters to manipulate them. Languages that use equations to calculate and move data *require* that each piece of data be named, even if it is a temporary variable for one-time use. This clogs up the name space, and it does so approximately in proportion to the size of the program in lines of code. [Dave-Wyland]

Defining Words

The Pearl of Forth!

virginia.edu

Copied from:
http://galileo.phys.virginia.edu/classes/551.jvn.fall01/primer.htm#create

Michael Ham has called the word pair CREATE…DOES>, the “pearl of Forth”.

- CREATE is a component of the compiler, whose function is to make a new dictionary entry with a given name.
- DOES> specifies a run-time action for the “child” words of a defining word.

Note

The point of a create does> word is that when you execute it, it produces a child word, and when that child word executes, it executes some code.

Defining “Defining” Words

CREATE finds its most important use in extending the powerful class of Forth words called “defining” words. The colon compiler “:” is such a word, as are VARIABLE and CONSTANT.

The definition of VARIABLE in high-level Forth is simple

```
: VARIABLE  CREATE   1 CELLS  ALLOT ;
```

We have already seen how VARIABLE is used in a program. An alternative definition found in some Forths is the variables are initialized to 0.

```
: VARIABLE  CREATE   0  ,  ;
```

Forth lets us define words initialized to contain specific values: for example, we might want to define the number 17 to be a word. CREATE and “,” (“comma”) can do this:

```
17 CREATE SEVENTEEN  ,  <cr>  ok
```

Now test it via

```
SEVENTEEN @ .  <cr>  17 ok .
```

Remarks:

- The word , (“comma”) puts TOS into the next cell of the dictionary and increments the dictionary pointer by that number of bytes.
- A word “C,” (“see-comma”) exists also — it puts a character into the next character-length slot of the dictionary and increments the pointer by 1 such slot. (In the ASCII character representation the slots are 1 byte long; Unicode characters require 2 bytes.)

Run-time vs. compile-time actions

In the preceding example, we were able to initialize the variable SEVENTEEN to 17 when we CREATEd it, but we still have to fetch it to the stack via SEVENTEEN @ whenever we want it. This is not quite what we had in mind. We would like to find 17 in TOS when SEVENTEEN is named. The word DOES> gives us the tool to do this.

The function of DOES> is to specify a run-time action for the “child” words of a defining word. Consider the defining word CONSTANT , defined in high-level (of course CONSTANT is usually defined in machine code for speed) Forth by

```
: CONSTANT  CREATE  ,  DOES>  @  ;
```

and used as

```
53 CONSTANT PRIME  <cr> ok
```

Now test it:

```
PRIME . <cr>  53  ok .
```

What is happening here?

- CREATE (hidden in CONSTANT) makes an entry named PRIME (the first word in the input stream following CONSTANT). Then “,” places the TOS (the number 53) in the next cell of the dictionary.
- Then DOES> (inside CONSTANT) appends the actions of all words between it and “;” (the end of the definition) —in this case, “@”— to the child word(s) defined by CONSTANT.

Dimensioned data (intrinsic units)

Here is an example of the power of defining words and of the distinction between compile-time and run-time behaviors.

Physical problems generally involve quantities that have dimensions, usually expressed as mass (M), length (L) and time (T) or products of powers of these. Sometimes there is more than one system of units in common use to describe the same phenomena.

For example, U.S. or English police reporting accidents might use inches, feet and yards; while Continental police would use centimeters and meters. Rather than write different versions of an accident analysis program it is simpler to write one program and make unit conversions part of the grammar. This is easy in Forth.

The simplest method is to keep all internal lengths in millimeters, say, and convert as follows:

```
: INCHES  254   10  */ ;
: FEET   [ 254 12 * ] LITERAL  10  */ ;
: YARDS  [ 254 36 * ] LITERAL  10  */ ;
: CENTIMETERS   10  * ;
: METERS   1000  * ;
```

Note

This example is based on integer arithmetic. The word */ means “multiply the third number on the stack by NOS, keeping double precision, and divide by TOS”. That is, the stack comment for */ is (a b c – a*b/c).

The usage would be

```
10 FEET  .  <cr>  3048 ok
```

The word “[” switches from compile mode to interpret mode while compiling. (If the system is interpreting it changes nothing.) The word “]” switches from interpret to compile mode.

Barring some error-checking, the “definition” of the colon compiler “:” is just

```
:  :   CREATE  ]  DOES>  doLIST  ;
```

and that of “;” is just

```
:  ;   next  [  ;  IMMEDIATE
```

Another use for these switches is to perform arithmetic at compile time rather than at run-time, both for program clarity and for easy modification, as we did in the first try at dimensioned data (that is, phrases such as

```
[ 254 12 * ] LITERAL
```

and

```
[ 254 36 * ] LITERAL
```

which allowed us to incorporate in a clear manner the number of tenths of millimeters in a foot or a yard.

The preceding method of dealing with units required unnecessarily many definitions and generated unnecessary code. A more compact approach uses a defining word, UNITS :

```
: D,  ( hi lo --)   SWAP  , ,  ;
: D@  ( adr -- hi lo)   DUP  @   SWAP   CELL+  @   ;
: UNITS  CREATE  D,   DOES> D@  */ ;
```

Then we could make the table

```
254 10          UNITS INCHES
254 12 *  10    UNITS FEET
254 36 *  10    UNITS YARDS
10  1           UNITS CENTIMETERS
1000  1         UNITS METERS

\ Usage:
10 FEET  . <cr>  3048  ok
3 METERS . <cr>  3000  ok
```

```
\ .......................
\ etc.
```

This is an improvement, but Forth permits a simple extension that allows conversion back to the input units, for use in output:

```
VARIABLE  <AS>    0 <AS> !
: AS      TRUE  <AS> ! ;
: ~AS     FALSE <AS> ! ;
: UNITS   CREATE  D,  DOES>  D@  <AS> @
          IF  SWAP  THEN
          */    ~AS  ;

\ UNIT DEFINITIONS REMAIN THE SAME.
\ Usage:
10 FEET  .   <cr>  3048  ok
3048 AS FEET  .  <cr>  10  ok
```

Advanced uses of the compiler

Suppose we have a series of push-buttons numbered 0-3, and a word WHAT to read them. That is, WHAT waits for input from a keypad: when button #3 is pushed, for example, WHAT leaves 3 on the stack.

We would like to define a word BUTTON to perform the action of pushing the n’th button, so we could just say:

```
WHAT BUTTON
```

In a conventional language BUTTON would look something like

```
: BUTTON  DUP  0 =  IF  RING  DROP  EXIT  THEN
          DUP  1 =  IF  OPEN  DROP  EXIT  THEN
          DUP  2 =  IF  LAUGH DROP  EXIT  THEN
          DUP  3 =  IF  CRY   DROP  EXIT  THEN
          ABORT" WRONG BUTTON!"   ;
```

That is, we would have to go through two decisions on the average.

Forth makes possible a much neater algorithm, involving a “jump table”. The mechanism by which Forth executes a subroutine is to feed its “execution token” (often an address, but not necessarily) to the word EXECUTE. If we have a table of execution tokens we need only look up the one corresponding to an index (offset into the table) fetch it to the stack and say EXECUTE.

One way to code this is

```
CREATE  BUTTONS  ' RING ,  ' OPEN ,  ' LAUGH ,  ' CRY ,

: BUTTON   ( nth --)    0 MAX  3 MIN
        CELLS  BUTTONS  +  @  EXECUTE  ;
```

Note how the phrase “0 MAX 3 MIN” protects against an out-of-range index. Although the Forth philosophy is not to slow the code with unnecessary error checking (because words are checked as they are defined), when programming a user interface some form of error handling is vital. It is usually easier to prevent errors as we just did, than to provide for recovery after they are made.

How does the action-table method work?

- CREATE BUTTONS makes a dictionary entry BUTTONS.
- The word ‘ (“tick”) finds the execution token (xt) of the following word, and the word , (“comma”) stores it in the data field of the new word BUTTONS. This is repeated until all the subroutines we want to select among have their xt’s stored in the table.
- The table BUTTONS now contains xt’s of the various actions of BUTTON.

- CELLS then multiplies the index by the appropriate number of bytes per cell, to get the offset into the table BUTTONS of the desired xt.
- BUTTONS + then adds the base address of BUTTONS to get the abso- lute address where the xt is stored.
- @ fetches the xt for EXECUTE to execute.
- EXECUTE then executes the word corresponding to the button pushed.

Simple!

If a program needs but one action table the preceding method suffices. However, more complex programs may require many such. In that case it may pay to set up a system for defining action tables, including both error-preventing code and the code that executes the proper choice. One way to code this is

```
: ;CASE   ;                     \ do-nothing word

: CASE:
    CREATE  HERE  -1  >R   0  ,   \ place for length
    BEGIN   BL  WORD  FIND        \ get next subroutine
       0=  IF   CR  COUNT  TYPE  ."  not found"  ABORT  THEN
       R>  1+  >R
       DUP  ,    ['] ;CASE  =
    UNTIL   R>   1-  SWAP  !      \ store length
    DOES>   DUP  @   ROT          ( -- base_adr len n)
            MIN  0  MAX           \ truncate index
            CELLS  +  CELL+  @  EXECUTE  ;
```

Note the two forms of error checking. At compile-time, CASE: aborts compilation of the new word if we ask it to point to an undefined subroutine:

```
case: test1   DUP  SWAP  X  ;case
X not found
```

and we count how many subroutines are in the table (including the do-nothing one, ;case) so that we can force the index to lie in the range [0,n].

```
CASE:  TEST  *  /  +  -  ;CASE  ok
15 3 0 TEST . 45  ok
15 3 1 TEST . 5  ok
15 3 2 TEST . 18  ok
15 3 3 TEST . 12  ok
15 3 4 TEST . . 3 15  ok
```

Just for a change of pace, here is another way to do it:

```
 : jtab:  ( Nmax --)      \ starts compilation
      CREATE              \ make a new dictionary entry
      1-  ,               \ store Nmax-1 in its body
 ;                        \ for bounds clipping

 : get_xt    ( n base_adr -- xt_addr)
      DUP  @      ( -- n base_adr Nmax-1)
      ROT         ( -- base_adr Nmax-1 n)
      MIN  0  MAX    \ bounds-clip for safety
      1+  CELLS+  ( -- xt_addr = base + 1_cell + offset)
 ;

 : |   '  ,   ;     \ get an xt and store it in next cell

 : ;jtab   DOES>  ( n base_adr --)   \ ends compilation
           get_xt  @  EXECUTE        \ get token and execute
it
 ;    \ appends table lookup & execute code

 \ Example:
 : Snickers   ." It's a Snickers Bar!"   ;   \ stub for test

 \ more stubs

 5 jtab:  CandyMachine
          |  Snickers
          |  Payday
          |  M&Ms
          |  Hershey
          |  AlmondJoy
 ;jtab
 3 CandyMachine  It's a Hershey Bar!   ok
 1 CandyMachine  It's a Payday!   ok
 7 CandyMachine  It's an Almond Joy!   ok
 0 CandyMachine  It's a Snickers Bar!   ok
-1 CandyMachine  It's a Snickers Bar!   ok
```

forth.org

Copied from http://forth.org/svfig/Len/definwds.htm

It has been said that one does not write a program in Forth. Rather, one extends Forth to make a new language specifically designed for the application at hand.

An important part of this process is the DEFINING WORD, by which it is possible to *combine a data structure with an action to create multiple instances that differ only in detail.*

The basics of create ... does>

Defining words are based on the Forth construct create ... does>, which beginners can apply mechanically. The steps are:

- Start a colon definition
- Write create
- Follow by words that lay down data or allot RAM, thus creating the body
- Write does>
- Follow by words that act on the body.

These steps are fairly simple, but understanding them is complex because there are three stages in the action of a defining word.

An example

Our example will be indexed-array, which allots an area of RAM. At run time, it takes an index, **i**, and returns the address of the **ith** cell. If **i=0** the address of the first cell is returned because Forth conventionally starts numbering at 0.

Stage 1

```
: indexed-array ( n -- ) ( i -- a )
   create cells allot
   does> swap cells +
;
```

Stage 2

```
20 indexed-array foo  \ Make a 1-dimensional array with 20
cells
```

Stage 3

```
3 foo                \ Put addr of fourth element on the
stack
```

Stage 1: Compiling the defining word

The first phase is in effect during the compilation of indexed-array, that is, between the colon and semicolon. The colon sets up a header. Then, execution tokens of ordinary Forth words are laid down, while Immediate Words are executed at once. The process is terminated by the semicolon.

The only Immediate word in indexed-array is “does>”. It lays down code that will act later in stage 2.

Stage 2: Creating a “child”

The second phase is in effect when “indexed-array” is used to create “foo”.

- create sets up a header
- cells allot reserves n cells, forming the “data field” (formerly called “body”) of foo.
- The code that was laid down by “does>” now comes into action. It changes the execution of “foo” so that it will:
    - Put the address of its data field on the stack, and then
    - Execute the Forth words between “does>” and the semicolon.

Stage 3: Executing the child

In the third phase, we execute “foo”.

- **i** is already on the stack, and the origin of the data field is put on top of that
- “swap” rearranges the stack
- “cells” multiplies **i** by the cell length
- “+” adds the result to the origin of the data field.

Warning

Important issues such as range checking and multi-dimensional arrays are not discussed here.

Note

Why is there a right angle-bracket in “does>”? It originated in early Forths in which create was followed by

“<builds … does>”. Later, the action of “<builds” was incorporated in “create”, but the spelling of “does>” was not changed.

Simple Example with LOOP

```
: gen-shc CREATE 5 0 DO  1 . LOOP DOES> @ ; ok.

gen-shc myc 1 1 1 1 1 ok.
```

Delay

Accurate Blocking Delay

This blocking delay is created using arm-none-eabi-as to produce the machine code which is then INLINED into a Forth Word.

Note

While the delay is being used, the CPU is not available to do anything else.

To see the same thing done using Interactive Assembly see Interactive Assembly Blocking Delay

Why do it this way ?

The reason that the millisecond blocking delay is done this way rather than in pure Forth is because it can be done in about half the binary code size compared to a pure Forth Word. Another reason is that sometimes maximum speed is required and this means using minimal instructions in the Forth Word. The delay example is a great way to demonstrate how to achieve this.

Note

The THUMB Instruction Set is used here

Using Pure Forth Instead

The pure Forth way is certainly much faster to write and debug compared to the alternate methods.

The “delay” Word below uses the common “do-loop”.

```
: delay
  0 do loop
;
```

Disassembly

```
see delay

2000068A: B500  push { lr }
2000068C: B430  push { r4  r5 }
2000068E: 2400  movs r4 #0
20000690: 0035  lsls r5 r6 #0
20000692: CF40  ldmia r7 { r6 }
20000694: 3401  adds r4 #1
20000696: 42AC  cmp r4 r5
20000698: D1FC  bne 20000694
2000069A: BC30  pop { r4  r5 }
2000069C: BD00  pop { pc }

Bytes: 20  ok.
```

Delay Measurments

Timing measurements were performed using the on chip Systick counter set to a count of 0.1ms using a Interrupt.

Input	Delay	Period	Comments
10000000	6.5359	seconds	uncalibrated

The Method below produces the following delays:

1. 105.2nS @ 48Mhz
2. 631.2nS @ 8Mhz

Note

Although very accurate due to the small loop, the accuracy will be affected by many factors such as the cache, interrupts and the stability of the clock. All blocking loop delays are affected by these things, but they are fine for general purpose delays of a known time period such a providing a minimum pulse width for a LCD display, blinking a LED and so on. If greater accuracy is required, consider using a Timer Peripheral instead.

In this example the delay Word is named “105ns” for use on a 48MHz system.

Delay Code

```
: 105ns  ( u --  )     \ 105.2ns delay @ 48mhz clock
  [
  $3e01 h,    \ subs   r6, #1
  $46c0 h,    \ nop
  $d1fc h,    \ bne.n  8 <loop>
  ] drop
;
```

Disassembly

```
see 105ns

20000688: B500  push { lr }
2000068A: 3E01  subs r6 #1
```

```
2000068C: 46C0  mov r8 r8
2000068E: D1FC  bne 2000068A
20000690: CF40  ldmia r7 { r6 }
20000692: BD00  pop { pc }

Bytes: 12  ok.
```

Delay Measurments

Timing measurements were performed using the on chip Systick counter set to a count of 0.1ms using a Interrupt.

This delay should work for a minium delay of 105.2 nS to a maximum delay of 451.86 seconds (7.53 Minutes)

Input	Delay	Period	Comments
1	105.2	ns	CALCULATED (too small to measure with systick)
1000	0.1052	ms	CALCULATED (too small to measure with systick)
1000000	0.1052	seconds	
10000000	1.0521	seconds	
$00FFFFFF	1.7651	seconds	
$0FFFFFFF	28.2415	seconds	

Input	Delay	Period	Comments
$FFFFFFFF	451.8640	seconds	(7.531067 minutes)

Millisecond Delay

The ms Word calls the 105ns Word and is scaled for 1 ms.

```
: ms ( u  -- )    \ for a 48 Mhz clock
  9505 * 105ns
;
```

Example usage

This lights the green LED on a Discovery board (clocked at 48MHz) for exactly 1 second.

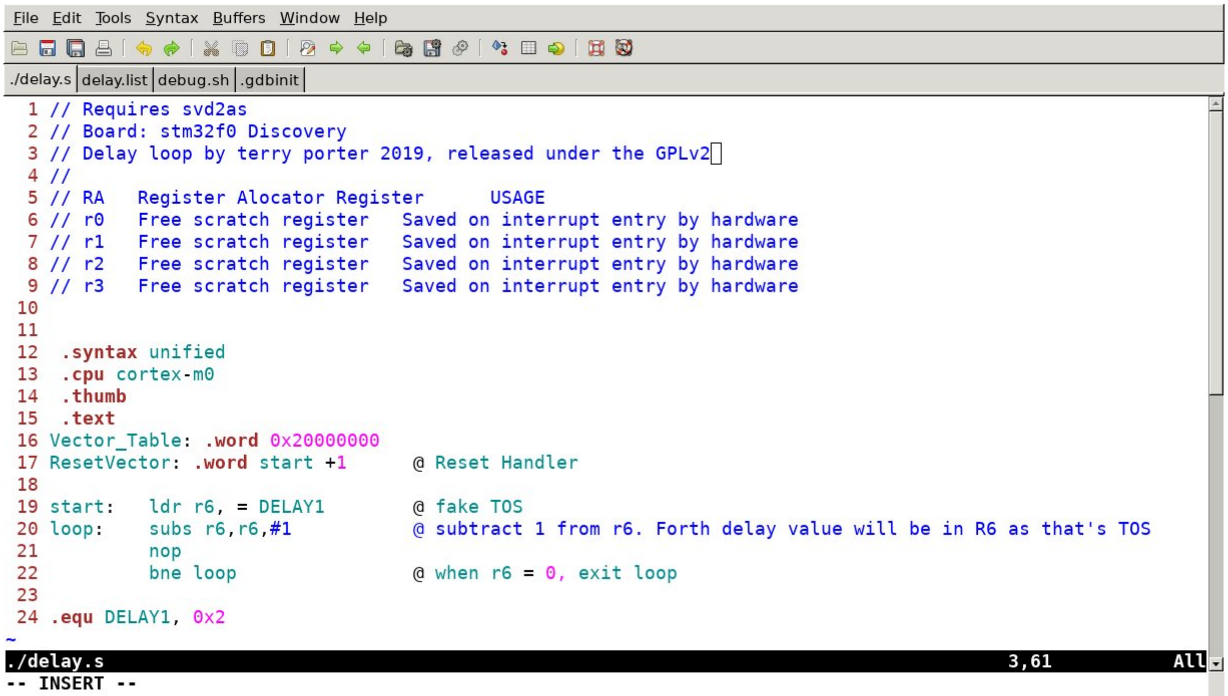
```
: green-led-on-1-second ( -- ) green-on 1000 ms green-off ;
```

How to Inline code

See also

Machine Code and Interactive Assembly

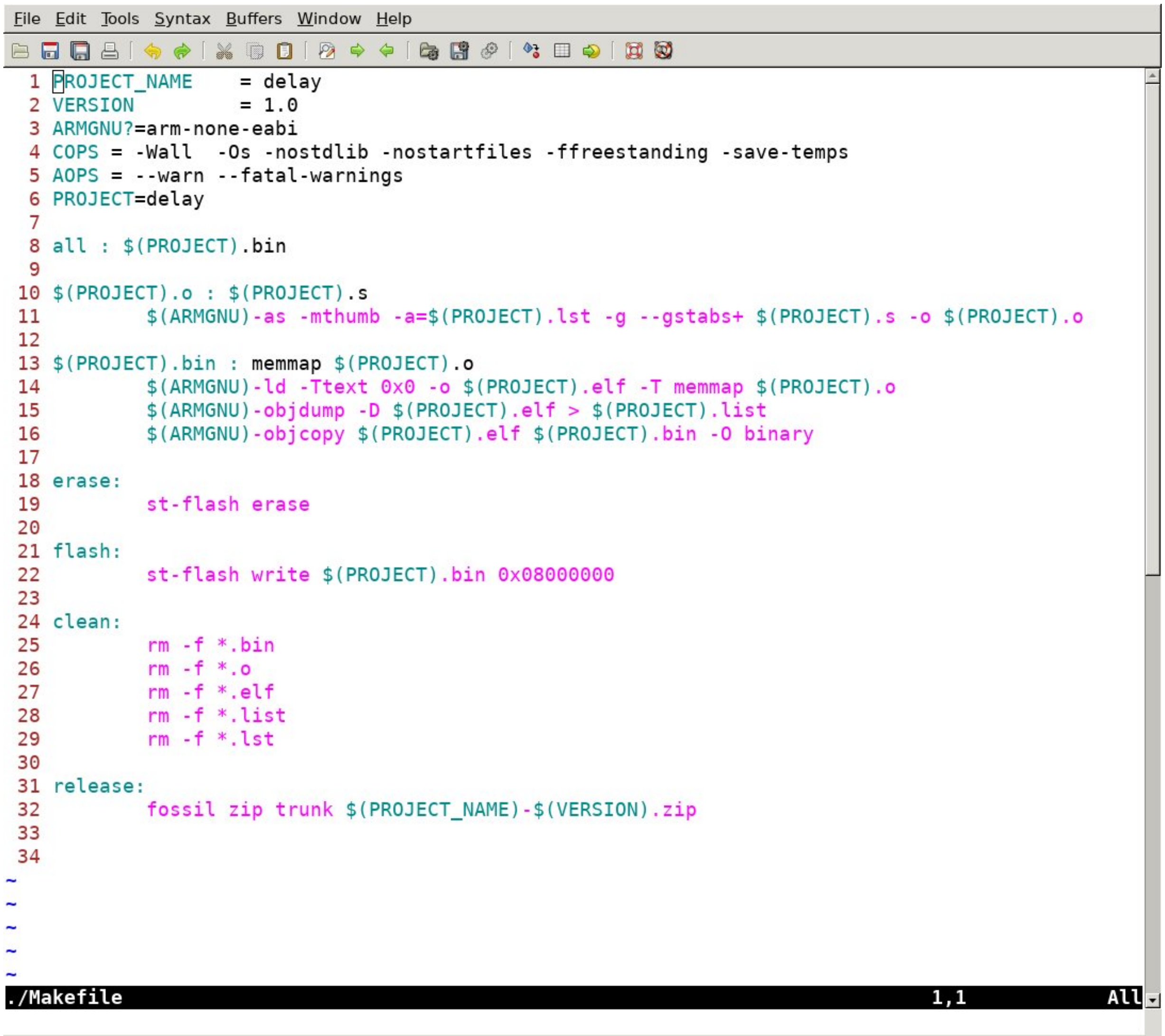
Write the code in Assembler


File
Edit
Tools
Syntax
Buffers
Window
Help
./delay.s
delay.list
debug.sh
.gdbinit

```
// Requires svd2as
// Board: stm32f0 Discovery
// Delay loop by terry porter 2019, released under the GPLv2
//
// RA   Register Alocator Register      USAGE
// r0   Free scratch register   Saved on interrupt entry by hardware
// r1   Free scratch register   Saved on interrupt entry by hardware
// r2   Free scratch register   Saved on interrupt entry by hardware
// r3   Free scratch register   Saved on interrupt entry by hardware


 .syntax unified
 .cpu cortex-m0
 .thumb
 .text
Vector_Table: .word 0x20000000
ResetVector: .word start +1      @ Reset Handler

start:   ldr r6, = DELAY1        @ fake TOS
loop:    subs r6,r6,#1           @ subtract 1 from r6. Forth delay value will be in R6 as that's TOS
         nop
         bne loop                @ when r6 = 0, exit loop

.equ DELAY1, 0x2
```

~
./delay.s
3,61
All
-- INSERT --

Compile the code with arm-none-eabi-as



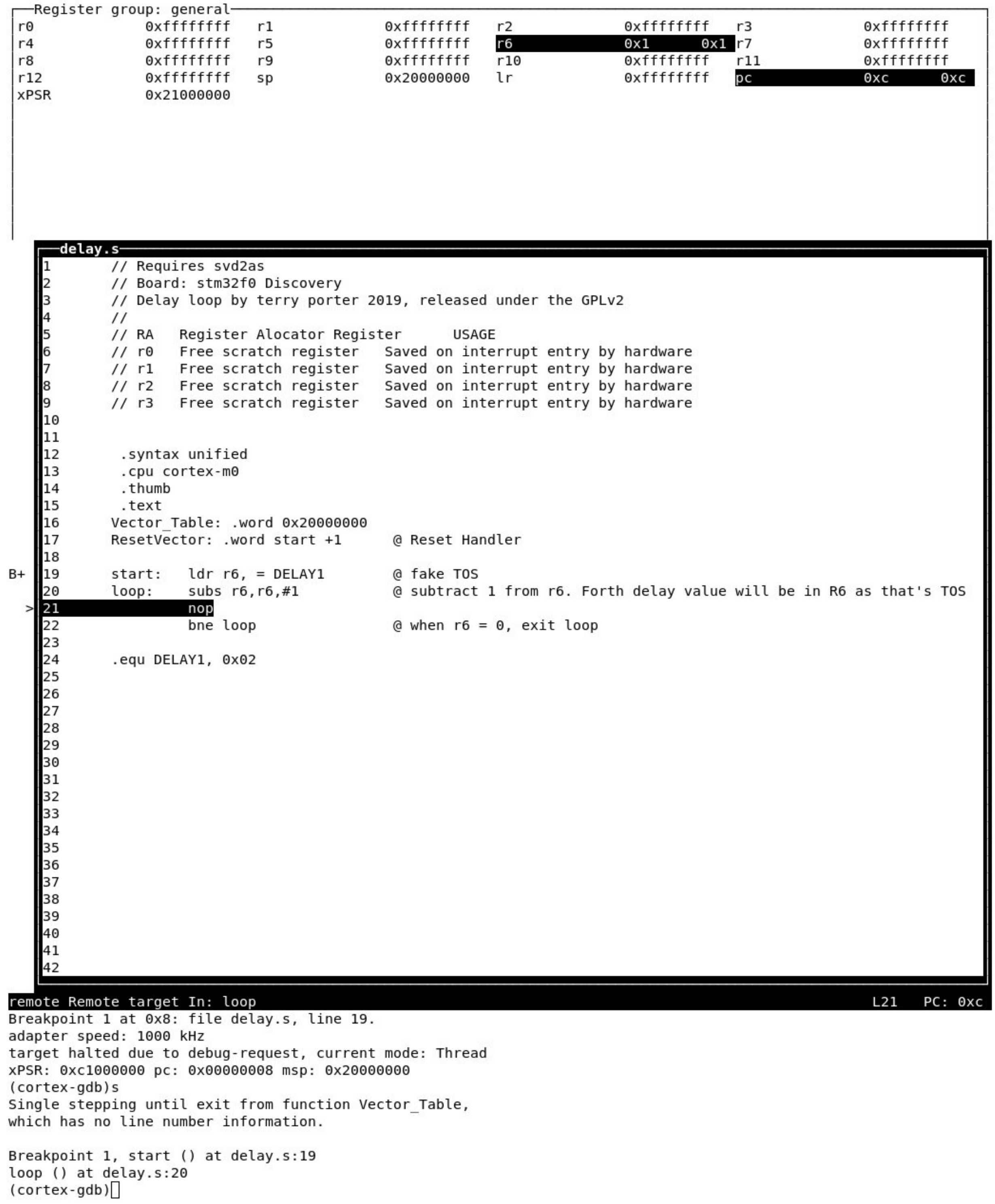
```
PROJECT_NAME    = delay
VERSION         = 1.0
ARMGNU?=arm-none-eabi
COPS = -Wall  -Os -nostdlib -nostartfiles -ffreestanding -save-temps
AOPS = --warn --fatal-warnings
PROJECT=delay

all : $(PROJECT).bin

$(PROJECT).o : $(PROJECT).s
        $(ARMGNU)-as -mthumb -a=$(PROJECT).lst -g --gstabs+ $(PROJECT).s -o $(PROJECT).o

$(PROJECT).bin : memmap $(PROJECT).o
        $(ARMGNU)-ld -Ttext 0x0 -o $(PROJECT).elf -T memmap $(PROJECT).o
        $(ARMGNU)-objdump -D $(PROJECT).elf > $(PROJECT).list
        $(ARMGNU)-objcopy $(PROJECT).elf $(PROJECT).bin -O binary

erase:
        st-flash erase

flash:
        st-flash write $(PROJECT).bin 0x08000000

clean:
        rm -f *.bin
        rm -f *.o
        rm -f *.elf
        rm -f *.list
        rm -f *.lst

release:
        fossil zip trunk $(PROJECT_NAME)-$(VERSION).zip


```

Test the Code in GDB

Note: the code shown here is slightly different to the actual “Delay Code” above because I have inserted a fake “Top Of Stack” line to supply R6 with a delay value of 0x2 for the debugging session. When this code is used with Forth, R6 will already contain a valid number from the user program so the fake TOS isn’t required.

GDB talks to the actual MCU via SWD and single steps thru the program *on the actual chip*, displaying register contents and following the program flow *on the actual chip*.

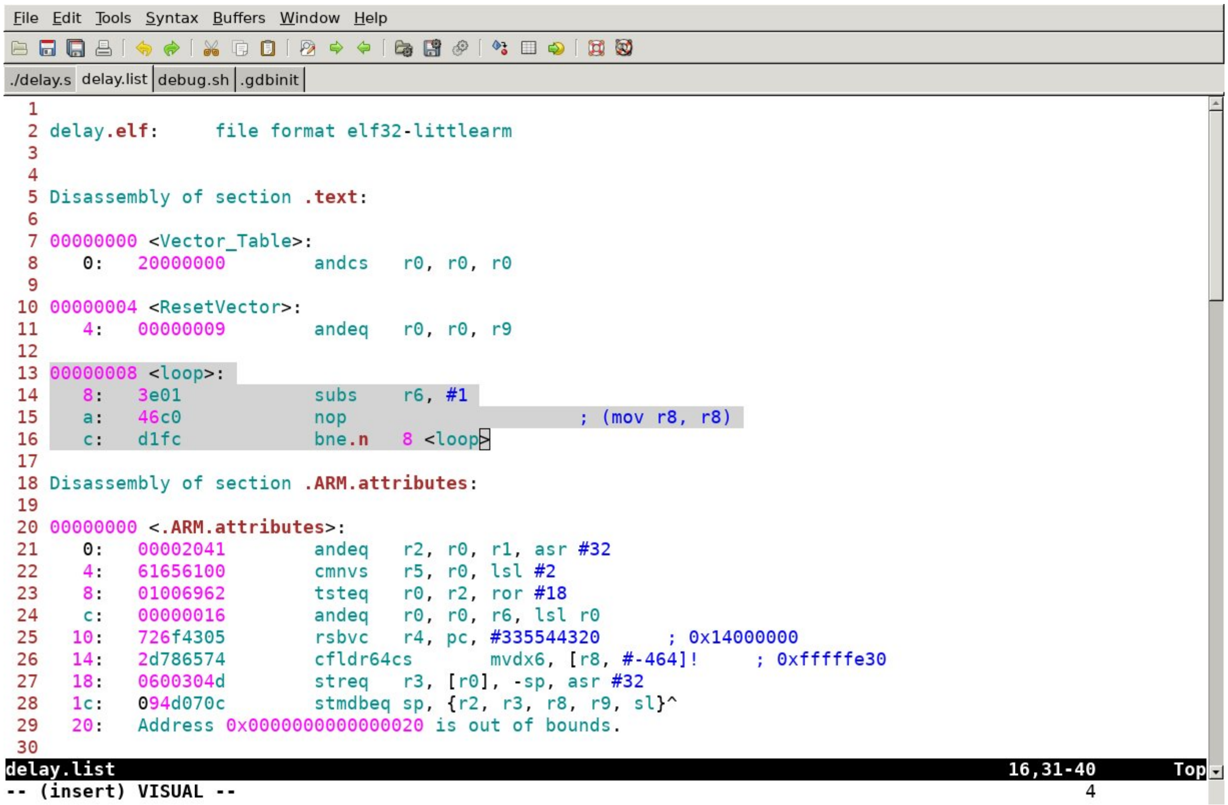
It’s incredible to think that this facility which once used to cost tens of *thousands* of dollars is available in a MCU costing $0.56 USD and that the GNU software is Free.



```
┌──Register group: general──────────────────────────────────────────────────────────────────────────────────┐
│r0             0xffffffff   r1             0xffffffff   r2             0xffffffff   r3             0xffffffff │
│r4             0xffffffff   r5             0xffffffff   r6             0x1      0x1 r7             0xffffffff │
│r8             0xffffffff   r9             0xffffffff   r10            0xffffffff   r11            0xffffffff │
│r12            0xffffffff   sp             0x20000000   lr             0xffffffff   pc             0xc      0xc │
│xPSR           0x21000000                                                                                   │
   ┌──delay.s─────────────────────────────────────────────────────────────────────────────────────────────┐
   │1       // Requires svd2as
   │2       // Board: stm32f0 Discovery
   │3       // Delay loop by terry porter 2019, released under the GPLv2
   │4       //
   │5       // RA   Register Alocator Register      USAGE
   │6       // r0   Free scratch register   Saved on interrupt entry by hardware
   │7       // r1   Free scratch register   Saved on interrupt entry by hardware
   │8       // r2   Free scratch register   Saved on interrupt entry by hardware
   │9       // r3   Free scratch register   Saved on interrupt entry by hardware
   │10
   │11
   │12       .syntax unified
   │13       .cpu cortex-m0
   │14       .thumb
   │15       .text
   │16      Vector_Table: .word 0x20000000
   │17      ResetVector: .word start +1      @ Reset Handler
   │18
B+ │19      start:   ldr r6, = DELAY1        @ fake TOS
   │20      loop:    subs r6,r6,#1           @ subtract 1 from r6. Forth delay value will be in R6 as that's TOS
  >│21               nop
   │22               bne loop                @ when r6 = 0, exit loop
   │23
   │24      .equ DELAY1, 0x02
   │25
   │26
   │27
   │28
   │29
   │30
   │31
   │32
   │33
   │34
   │35
   │36
   │37
   │38
   │39
   │40
   │41
   │42
   └──────────────────────────────────────────────────────────────────────────────────────────────────────┘
remote Remote target In: loop                                                                L21   PC: 0xc
Breakpoint 1 at 0x8: file delay.s, line 19.
adapter speed: 1000 kHz
target halted due to debug-request, current mode: Thread
xPSR: 0xc1000000 pc: 0x00000008 msp: 0x20000000
(cortex-gdb)s
Single stepping until exit from function Vector_Table,
which has no line number information.

Breakpoint 1, start () at delay.s:19
loop () at delay.s:20
(cortex-gdb)
```

Cut and Paste the Machine Code into a Forth Word

The paste must be edited to suit Forth as seen in the [“Delay Code”]() above.



```

delay.elf:     file format elf32-littlearm


Disassembly of section .text:

00000000 <Vector_Table>:
   0:   20000000        andcs   r0, r0, r0

00000004 <ResetVector>:
   4:   00000009        andeq   r0, r0, r9

00000008 <loop>:
   8:   3e01            subs    r6, #1
   a:   46c0            nop                     ; (mov r8, r8)
   c:   d1fc            bne.n   8 <loop>

Disassembly of section .ARM.attributes:

00000000 <.ARM.attributes>:
   0:   00002041        andeq   r2, r0, r1, asr #32
   4:   61656100        cmnvs   r5, r0, lsl #2
   8:   01006962        tsteq   r0, r2, ror #18
   c:   00000016        andeq   r0, r0, r6, lsl r0
  10:   726f4305        rsbvc   r4, pc, #335544320      ; 0x14000000
  14:   2d786574        cfldr64cs       mvdx6, [r8, #-464]!     ; 0xfffffe30
  18:   0600304d        streq   r3, [r0], -sp, asr #32
  1c:   094d070c        stmdbeq sp, {r2, r3, r8, r9, sl}^
  20:   Address 0x0000000000000020 is out of bounds.

```

```
delay.list                                                        16,31-40       Top
-- (insert) VISUAL --                                                     4
```

FORTH DEVELOPMENT ENVIRONMENT

I believe that Forth like any other programming language benefits from a well organised project development environment.

This article is *a work in progress* because it’s using Open Software on the PC allowing me to continually improve my designs.

Trqaditional Forth

Traditional Forth uses a serial terminal to interactively command (in real time) and upload source to the MCU. The Forth I use with Cortex-M is a traditional but modern Forth.

Traditional Forth serial terminal use has a few variations not limited to my description below because anything is possible with Forth.

1. A terminal such as Picocom, Minicom even Hyperterm in Windows is used to interactively command in real time and to upload source to the MCU making use of the ASCII serial upload capability of the terminal or a ‘helper’ program called by the terminal for uploading. In these cases the source code comments are read but rejected by the on-mcu Forth compiler and End Of Line delays are inserted by the terminal (where supported) to prevent the on-mcu Forth compiler choking on long or complex source. There is usually no handshaking of any kind to use as a alternative to the EOL delays and

speeds are often 9600 Baud so that errors and warnings from the on-mcu Forth compiler can be read by the programmer as they occur.

Advantages

- simple

Disadvantages

- unbearably slow

2. A purpose made terminal with on board smarts and file upload capability such as eForthcom is used. This special terminal is made especially to suit Forth, and strips the comments from the source, recognizes errors, highlights them in color and stops the upload completely at the error. eForthcom works by waiting for the “OK.” return from the on-MCU Forth compiler giving the fastest possible upload speed without EOL delays or traditional hardware handshaking.

Advantages

- quick and easy
- very fast
- limited to a maximum speed of 115,200 Baud
- configurable error detection for a predefined choice of different Forths
- coloured error messages
- customisable error handling
- halts on errors
- written in Forth and bugfree
- actively maintained by the developer Manfred Mahlow

Disadvantages

- Linux only binary distribution (I use FreeBSD)

My Development System

Gnu Screen is the key component of my system and nothing else I’m aware of comes close. Not Tmux, not CU, nothing.

Essential Development Capability

Only Gnu Screen satisfies all my requirements:

Requirement	**Because ?**
a serial terminal	talk Forth to the MCU
hardware handshaking	maximise the speed of source code uploads to the MCU
controllable logging	acquire MCU code; automated cloning of MCU Flash image
remote named terminal sessions	automated script control of the terminal
remote commands	makefile control of source code uploads to MCU
remote uploads	makefile control of source code uploads to MCU

Requirement	Because ?
at least one sub process	Forth on-chip compiler error message detection (via SED) and terminal bell alert on errors

Key Aspects of the development system

a. Interactively command the on-mcu Forth in the gnu screen window.
b. upload source to the mcu when I click my editor ‘make button’. It does this by opening a remote connection to gnu screen and uploading commands and source via a Makefile. I do all this from the editor window but I see the upload as it occurs in the gnu screen window … sort off, because at 460800 baud with hardware handshaking its hard to read anything as it flashes up the screen.
c. all comments are stripped from the source by SED before being uploaded in b. above
d. Any errors from the on-mcu Forth compiler are highlighted in RED in gnu screen using a sub-process involving configurable SED keywords and ANSII escape codes inserted into the RX stream by the MCU. Warnings are coloured BLUE. Warnings and errors RING the gnu screen terminal bell as they occur. This is required because the uploads are to0 fast to read.
e. Stop the upload at the error if desired. This can be done with a toggle switch at the development hardware. I don’t use this nowadays as color errors and the beep of the terminal bell are all I need. Gnu screen has a long history buffer and I can easily

scroll back to look for the colored error after hearing a beep.

f. Suck the entire Dictionary from the MCU in IntelHex and build a complete binary image of everything including user Words from a project. This image can then be flashed to another mcu which is then a 100% clone.

Personally I find traditional Forth such as 1. above far,far,far too slow and I wouldn’t blame anyone who tries Forth in this way from thinking it was a load of ancient crap compared to using arm.none.eabi and GDB via SWD on a Cortex-M.

I developed my system until it was faster than arm.none.eabi and GDB via SWD on a Cortex-M using the usual write code, try, fix, repeat development cycle.

My system is far more than just gnu screen above as it involves:

1. a project builder
2. integrated Fossil SCM with webserver
3. integrated configurable CMSIS-SVD memory map and bitfield generator for any STM32 Cortex-M mcu. This means I can easily write Forth code as I have every peripheral memory mapped automatically. I never have to refer to the documentation for memory locations or register bitfields. Doing this manually is tedious, error prone and incredibly slow. No one should ever have to do this and I’m amazed by Forth people who still do this manually on Cortex-M.

Overall Operation

1. Create project directory
2. Cd to project directory and type “project-builder”, which fills the directory with project templates and

   stubs
3. Edit project source code
4. Run Make which:
    1. Resets the target MCU
    2. Strips all comments from the Forth source
    3. Uploads the Forth source to the target at 460800 Baud with hardware handshaking
5. Test target
6. If correct target operation observed, go to 7. else go to to 3. and repeat
7. Finished. Run “clone.sh” and create a MD5 named executable project binary for the target.

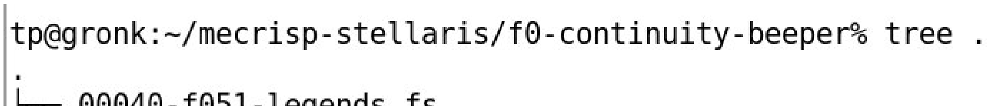
PROJECT BUILDER

Any Forth project is going to need a well structured directory containing quite a few files and doing this by hand for every project is error prone and incredibly tedious, in fact it’s just not reasonable.

In my case I built a “project builder” shell script (currently about 800 lines) which generates project stubs automatically. This system is somewhat similar to CrystalScad

Basically I just create a directory named for the new project, then in that directory type the name of my script and *everything* is auto created in about 0.1 seconds.

This picture shows the directory contents of a brand new, automatically pre-populated project



```
tp@gronk:~/mecrisp-stellaris/f0-continuity-beeper% tree .
.
└── 00040-f051-legends.fs
```

```
├── 00040-f051-legends.fs
├── Makefile
├── STM32F0xx-tp1.svd
├── STM32F0xx-tp1.svd-cleaned.svd
├── STM32F0xx-tp1.svd-cleaned.svd.uf.svd
├── bitfields.fs
├── bitfields.xsl
├── clone.flash.sh
├── clone.sh
├── edit-all-files-f0-continuity-beeper.sh
├── edit-f0-continuity-beeper.fs.sh
├── edit-f0-continuity-beeper.md.sh
├── edit-f0-continuity-beeper.sch.sh
├── edit-template.xml.sh
├── f0-continuity-beeper.error.bell.sh
├── f0-continuity-beeper.fs
├── f0-continuity-beeper.includes.fs
├── flowcharts
│   └── flowchart.readme
├── fossil-ui-start.sh
├── gafrc
├── memmap.fs
├── misc-files
├── mk.template.xsl
├── pdf
├── pics
├── raw-svd-clean.sh
├── rcas.sh
├── schematics
│   └── f0-continuity-beeper.sch
├── screen-f0-continuity-beeper-cuaU0-460800-ctsrts.sh
├── screen-f0-continuity-beeper-cuaU1-460800-ctsrts.sh
├── screen-f0-continuity-beeper-cuaU2-460800-ctsrts.sh
├── sphinx
│   └── sphinx.readme
├── svdcutter.xsl
├── svduf.xsl
├── template.xml
├── text
├── upload.main.sh
├── upload.utils.sh
├── www
│   └── f0-continuity-beeper.md
└── z-WARNING-DESTROY-f0-continuity-beeper.fossil.sh

8 directories, 36 files
```

Project Builder Highlights

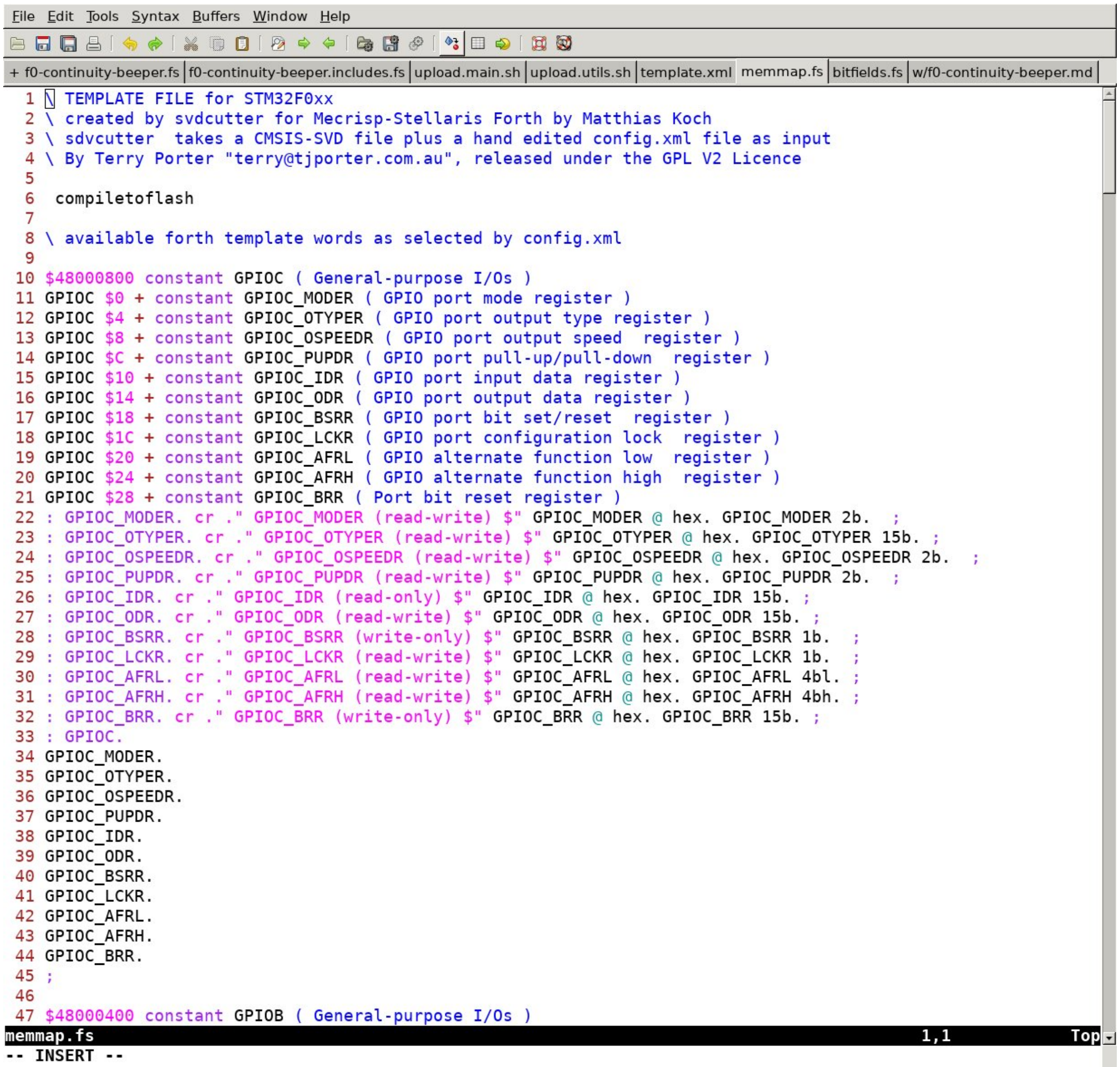
- Makefile creation
- Code Versioning System: set up and running, with the initial project files already committed. I use Fossil SCM for this.
- Schematic Capture: with a various templates ready for me to begin testing and documenting. Usually I start with a STM32F0 Discovery Board, so that’s my main schematic template. gEDA is what I use.
- Directories: ready to receive files as I create them, i.e. pdf, flowchart, pics, text, misc
- A library symlink to my versioned STM32F Mecrisp-Stellaris library so I can pull them in as needed
- Py.Sphinx documentation system: (py.sphinx is used to create this site) ready to start adding documents.
- Editor: which will open with several files ready to edit, pre-created and pre-named by the *project builder* shell script.
- Serial Terminal
- Svd2forth files to suit the MCU. These are used to generate Memory Mapped Register words, preloaded before the actual project files and controlled by a Makefile.
- Utility applications such as cloning the project source into a executable target binary.
- Finally, a *z-WARNING-DESTROY-test-project.fossil.sh* shell script that will remove every project file, the Fossil repository and finally the project directory itself. This has a “are you sure” confirmation before proceeding. Once a project is underway and I’m happy with it, I may manually delete the *z-WARNING-DESTROY-test-project.fossil.sh* shell script just to be on the safe side. It’s saved in the Fossil Code Repo anyway if I need it later.

The *project builder* shell script also refuses to dump all these files into a non-empty directory.

It’s all done with one SH script, no Java, no Eclipse, no delay.

In fact, the above system is perfectly responsive on a Raspberry Pi V3 with no delays. Try running the free Atollic TrueStudio for STM32 on a RPI ? I bet it takes a week just to load, and that’s not including CubeMX.

This picture shows the auto opened editor files

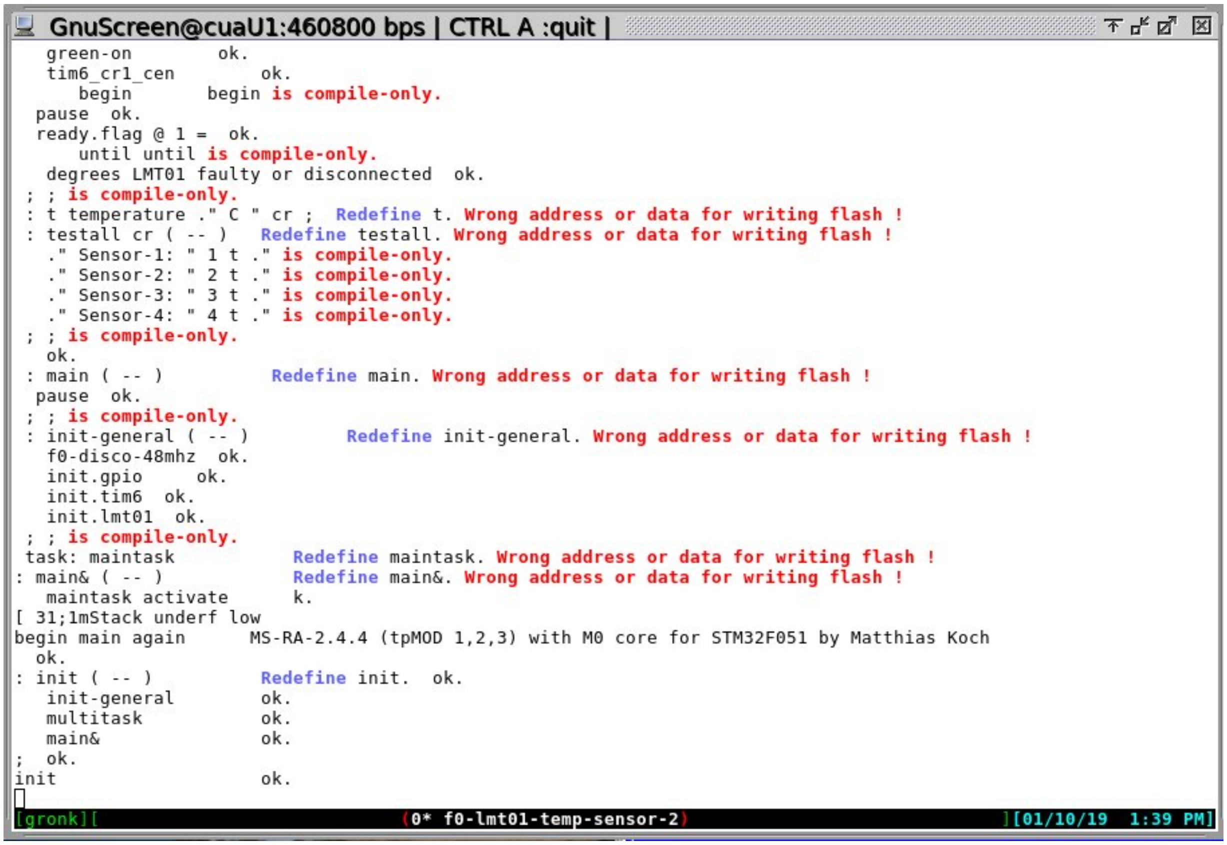

File Edit Tools Syntax Buffers Window Help
+ f0-continuity-beeper.fs
f0-continuity-beeper.includes.fs
upload.main.sh
upload.utils.sh
template.xml
memmap.fs
bitfields.fs
w/f0-continuity-beeper.md

```
\ TEMPLATE FILE for STM32F0xx
\ created by svdcutter for Mecrisp-Stellaris Forth by Matthias Koch
\ sdvcutter  takes a CMSIS-SVD file plus a hand edited config.xml file as input
\ By Terry Porter "terry@tjporter.com.au", released under the GPL V2 Licence

 compiletoflash

\ available forth template words as selected by config.xml

$48000800 constant GPIOC ( General-purpose I/Os )
GPIOC $0 + constant GPIOC_MODER ( GPIO port mode register )
GPIOC $4 + constant GPIOC_OTYPER ( GPIO port output type register )
GPIOC $8 + constant GPIOC_OSPEEDR ( GPIO port output speed  register )
GPIOC $C + constant GPIOC_PUPDR ( GPIO port pull-up/pull-down  register )
GPIOC $10 + constant GPIOC_IDR ( GPIO port input data register )
GPIOC $14 + constant GPIOC_ODR ( GPIO port output data register )
GPIOC $18 + constant GPIOC_BSRR ( GPIO port bit set/reset  register )
GPIOC $1C + constant GPIOC_LCKR ( GPIO port configuration lock  register )
GPIOC $20 + constant GPIOC_AFRL ( GPIO alternate function low  register )
GPIOC $24 + constant GPIOC_AFRH ( GPIO alternate function high  register )
GPIOC $28 + constant GPIOC_BRR ( Port bit reset register )
: GPIOC_MODER. cr ." GPIOC_MODER (read-write) $" GPIOC_MODER @ hex. GPIOC_MODER 2b.  ;
: GPIOC_OTYPER. cr ." GPIOC_OTYPER (read-write) $" GPIOC_OTYPER @ hex. GPIOC_OTYPER 15b. ;
: GPIOC_OSPEEDR. cr ." GPIOC_OSPEEDR (read-write) $" GPIOC_OSPEEDR @ hex. GPIOC_OSPEEDR 2b.  ;
: GPIOC_PUPDR. cr ." GPIOC_PUPDR (read-write) $" GPIOC_PUPDR @ hex. GPIOC_PUPDR 2b.  ;
: GPIOC_IDR. cr ." GPIOC_IDR (read-only) $" GPIOC_IDR @ hex. GPIOC_IDR 15b. ;
: GPIOC_ODR. cr ." GPIOC_ODR (read-write) $" GPIOC_ODR @ hex. GPIOC_ODR 15b. ;
: GPIOC_BSRR. cr ." GPIOC_BSRR (write-only) $" GPIOC_BSRR @ hex. GPIOC_BSRR 1b.  ;
: GPIOC_LCKR. cr ." GPIOC_LCKR (read-write) $" GPIOC_LCKR @ hex. GPIOC_LCKR 1b.  ;
: GPIOC_AFRL. cr ." GPIOC_AFRL (read-write) $" GPIOC_AFRL @ hex. GPIOC_AFRL 4bl. ;
: GPIOC_AFRH. cr ." GPIOC_AFRH (read-write) $" GPIOC_AFRH @ hex. GPIOC_AFRH 4bh. ;
: GPIOC_BRR. cr ." GPIOC_BRR (write-only) $" GPIOC_BRR @ hex. GPIOC_BRR 15b. ;
: GPIOC.
GPIOC_MODER.
GPIOC_OTYPER.
GPIOC_OSPEEDR.
GPIOC_PUPDR.
GPIOC_IDR.
GPIOC_ODR.
GPIOC_BSRR.
GPIOC_LCKR.
GPIOC_AFRL.
GPIOC_AFRH.
GPIOC_BRR.
;

$48000400 constant GPIOB ( General-purpose I/Os )
```

SERIAL TERMINAL

https://www.gnu.org/software/screen/ showing coloured error messages from the MCU on chip Forth compiler.

This picture shows various contrived error and warning messages in colour



```
   green-on        ok.
   tim6_cr1_cen        ok.
      begin       begin is compile-only.
  pause  ok.
  ready.flag @ 1 =  ok.
      until until is compile-only.
   degrees LMT01 faulty or disconnected  ok.
 ; ; is compile-only.
 : t temperature ." C " cr ;  Redefine t. Wrong address or data for writing flash !
 : testall cr ( -- )   Redefine testall. Wrong address or data for writing flash !
   ." Sensor-1: " 1 t ." is compile-only.
   ." Sensor-2: " 2 t ." is compile-only.
   ." Sensor-3: " 3 t ." is compile-only.
   ." Sensor-4: " 4 t ." is compile-only.
 ; ; is compile-only.
   ok.
 : main ( -- )          Redefine main. Wrong address or data for writing flash !
  pause  ok.
 ; ; is compile-only.
 : init-general ( -- )         Redefine init-general. Wrong address or data for writing flash !
   f0-disco-48mhz  ok.
   init.gpio     ok.
   init.tim6  ok.
   init.lmt01  ok.
 ; ; is compile-only.
 task: maintask          Redefine maintask. Wrong address or data for writing flash !
: main& ( -- )           Redefine main&. Wrong address or data for writing flash !
   maintask activate     k.
[ 31;1mStack underf low
begin main again      MS-RA-2.4.4 (tpMOD 1,2,3) with M0 core for STM32F051 by Matthias Koch
  ok.
: init ( -- )          Redefine init.  ok.
   init-general        ok.
   multitask           ok.
   main&               ok.
;  ok.
init                   ok.

[gronk][                         (0* f0-lmt01-temp-sensor-2)                            ][01/10/19  1:39 PM]
```

I edit Forth Code in a GUI editor then click on the editor “make’ icon to test the code on the target. This strips all comments from the source,and uploads the code at 460800 Baud with RTS handshaking to the target MCU which then runs the code.

The PC ‘bell’ also emits a loud “beep” when ever a error is reported on the terminal. It does this by parsing all received data thru Sed where error keywords ring the terminal bell.

This method is the fasted I’ve found to date. When a large and complex Word is encountered the GNU Screen Window freezes for a instant while the target MCU compiles it then resumes the upload scrolling code on the terminal screen faster than I can read it.

For years I used a few dedicated Serial Terminals, the best being e4thcom but only Gnu Screen has allowed the freedom to implement new features as they occur to me.

Features

1. Allows the editor to perform uploads with a ‘Make’ button click
2. Colour error highlighting
3. Speed of 460800 Baud to the target
4. PC bell beeps on configurable target errors
5. Comments automatically removed *before* upload to the target.

Frankly if one can’t detect code errors automatically during a *fast* source upload, Mecrisp-Stellaris is going to be a very disappointing experience ihmo. I believe watching a large Forth program upload slowly just so errors can be observed is error prone, painfully slow and outdated compared to modern C development systems.

In my experience this system is faster than using C and GDB on the same Intel i7 workstation.

CrystalScad

https://github.com/jglauche/CrystalScad

CrystalSCAD comes with a generator that generates project stubs automatically for you. Run this command from a terminal in the directory that you want to create a project: # crystalgen new [my_project_name]

Change [my_project_name] to the name of your project

A project named “my_project” will create those files and directories:

```
my_project/my_project.rb - the controller
my_project/lib/assemblies - for putting together assemblies
of individual parts
my_project/lib/electronics - put electronics here
my_project/lib/hardware - put hardware parts in here
my_project/lib/printed - put parts that you want to print in
here
my_project/lib/lasercut - put sheets that need to be cut (by
laser or other) in here
my_project/lib/assemblies/my_project_assembly.rb - dummy
assembly
my_project/lib/printed/testcube.rb - dummy printed part
my_project/my_project.observr - observer file
```

Open up the controller (here my_project/my_project.rb) in the text editor of your choice. It contains the information on how to start the observer.

Development Tips

Just a few handy debug methods I’ve developed along the way …

PR: Monitor Stack Values and Program Flow

Forth is definitely ALL about the Stack and visualising the Stack to design code has been, and continues to be my main challenge.

This Stack visualisation method is *really simple*, yet I use it all the time when designing code and fixing bugs. Used with VIM Mapping, it’s fast and easy to use. I’m still a Forth beginner and I need all the help I can get, so being able to rapidly insert print statements to follow the stack and program flow, plus show nothing is left on the stack has made Forth a lot easier for me.

```
: pr ( identifier -- identifier print stack ) . .s cr ;
\ A debugging stack printer
```

Vim Key Mapping

```
" insert "<linenumber> pr" only in INSERT MODE for Forth debugging
map! _P <C-R>=printf("%d pr", line('.'))<CR>
```

In Insert mode I move the cursor to the source code line I want the stack printed then I use the key sequence ” _p” and Vim inserts the line number followed by “pr” as in the example below.

PR: Usage Example

```
\ Program Name: pr-example.fs  for Mecrisp-Stellaris by Matthias Koch and licensed under the GPL
\ This program: 'pr' is a debugging 'print statement'
\ Hardware: STM32F0 Discovery board
\ Copyright 2017 t.porter <terry@tjporter.com.au> and licensed under the GPL


\ This is the very simple "pr" program. I use it with a VIM map which inserts the program line number before the "pr" as seen in the example below.
 : pr . .s cr ;  \ print the stack and follow program flow for debugging

\ Example program showing "pr" stack printing interspersed within the "pr-example" program

 :  pr-example cr
   1 pr
     4 0 do
   2 pr
     I
   3 pr
      dup dup
   4 pr
     ." Loop#" . cr
   5 pr
     +
   6 pr
     drop
     loop
   7 pr
 ;


\ ...................... screenpic .....................
\   pr-example
\
\   1 [0 ] 0
\   2 [0 ] 0
\   3 [1 ] 0 0
\   4 [3 ] 0 0 0 0
\   Loop#0
\   5 [2 ] 0 0 0
\   6 [1 ] 0 0
\   2 [0 ] 0
\   3 [1 ] 0 1
\   4 [3 ] 0 1 1 1
\   Loop#1
\   5 [2 ] 0 1 1
\   6 [1 ] 0 2
\   2 [0 ] 0
\   3 [1 ] 0 2
\   4 [3 ] 0 2 2 2
\   Loop#2
\   5 [2 ] 0 2 2
\   6 [1 ] 0 4
\   2 [0 ] 0
\   3 [1 ] 0 3
\   4 [3 ] 0 3 3 3
\   Loop#3
\   5 [2 ] 0 3 3
\   6 [1 ] 0 6
\   7 [0 ] 0
\    ok.
\ ...................... screenpic .....................
```

Balancing The Return Stack

Forth does have a ‘Local Variable’, it’s called The ‘Return Stack’, but it must be balanced or your program will crash.

Note

“Balancing the Return Stack” means making sure that the number of >R equals the number of R> within the same definition. Special care is needed with looping.

Why use the Return Stack to as a Local Variable, why not just use a Global Variable ?

1. The example Split program below saves 24 Bytes over Split2 and in Embedded, memory is a finite resource and costly to increase.
2. I prefer Words to be self contained if possible
3. The challenge of not using Global Variables

Warning

It’s often convenient to be able to put a stack item temporarily in the return stack, but the following rules must be observed because if the Return Stack isn’t balanced, this is *very bad* and if it doesn’t crash now, it will cause problems later.

- Data put on the return stack must be taken back within the same word.
- Data put on the return stack outside a ?DO (DO) … LOOP (+LOOP) cannot be accessed within the loop.
- Data put on the return stack within a ?DO (DO) … LOOP (+LOOP) must be taken back before leaving the loop.
- Data cannot be on the return stack when executing I or J in a loop.

Split Program

This program splits a number into digits, and utilises the Return Stack as a local variable.

```
: split
   >R                     \ Save the input on the Return
Stack
   BEGIN
       R> ?DUP 0 > WHILE   \ Fetch from the Return Stack,
duplicate because " 0 >" and "/MOD" both use it, then
continue WHILE greater than zero. ?DUP is used instead of DUP
to prevent a zero being left on the stack after the program
has completed.
       10 /MOD >R . cr     \ divide by 10 and save the result
on the Return Stack, print each remainder on a new line
   REPEAT
;

\ ------- Split program output ---------- \
123456789 split

9
8
7
6
5
4
3
2
1
```

Note

At first glance, the Return Stack doesn’t look balanced does it ?

How To Determine if The Return Stack is Balanced

In this case, I used a variable (rb) to tally my Return Stack *pushes* and *pulls*. A ‘rb’ total of 0 shows the Return Stack is balanced.

Return Stack Operation	code	Description

Return Stack Operation	**code**	**Description**
Push	>R rb @ 1+ rb !	Add one to ‘rb’
Pull	R> rb @ 1- rb !	Subtract one from ‘rb’

Return Stack Accounting

Every Return Stack ADD is worth 1 and REMOVE is worth -1. A total of 0 shows the Return Stack is balanced, so no visual comparison is required.

```
0 variable rb

: split-rbalance cr
>R rb @ 1+ rb !
BEGIN
  R> rb @ 1- rb !
  ?DUP 0 > WHILE
  10 /MOD
  >R rb @ 1+ rb !
  .  cr
REPEAT
cr
rb @ 0= if
."  The Return Stack is balanced " cr cr
else
."  UNBALANCED ! The Return Stack is NOT balanced! rb =  " rb
@ . cr cr
then
;
```

Return Stack Accounting Output

```
123456789 split-rbalance

 9
 8
 7
 6
 5
 4
 3
 2
 1

 The Return Stack is balanced

ok.
```

Compare Local Variable vs Return Stack code size

Dissasemble Split

Split is 64 bytes

```
see split

20000398: B500  push { lr }
2000039A: B440  push { r6 }
2000039C: CF40  ldmia r7 { r6 }
2000039E: BC08  pop { r3 }
200003A0: 3F04  subs r7 #4
200003A2: 603E  str r6 [ r7 #0 ]
200003A4: 461E  mov r6 r3
200003A6: 2E00  cmp r6 #0
200003A8: D001  beq 200003AE
200003AA: 3F04  subs r7 #4
200003AC: 603E  str r6 [ r7 #0 ]
200003AE: 2E00  cmp r6 #0
200003B0: CF40  ldmia r7 { r6 }
200003B2: DD11  ble 200003D8
200003B4: 3F04  subs r7 #4
200003B6: 603E  str r6 [ r7 #0 ]
200003B8: 260A  movs r6 #A
```

```
200003BA: 2082  movs r0 #82
200003BC: 0180  lsls r0 r0 #6
200003BE: 3039  adds r0 #39
200003C0: 4780  blx r0  --> /mod
200003C2: B440  push { r6 }
200003C4: CF40  ldmia r7 { r6 }
200003C6: 2086  movs r0 #86
200003C8: 01C0  lsls r0 r0 #7
200003CA: 3021  adds r0 #21
200003CC: 4780  blx r0  --> .
200003CE: 2097  movs r0 #97
200003D0: 0180  lsls r0 r0 #6
200003D2: 3033  adds r0 #33
200003D4: 4780  blx r0  --> cr
200003D6: E7E2  b 2000039E
200003D8: BD00  pop { pc }
```

Alternative Split2 Progran using Variable

```
0 variable split-1

: split2
    split-1 !
    BEGIN
        split-1 @ ?DUP 0 > WHILE
        10 /MOD split-1 ! . cr
   REPEAT
;
```

Dissasemble Split2

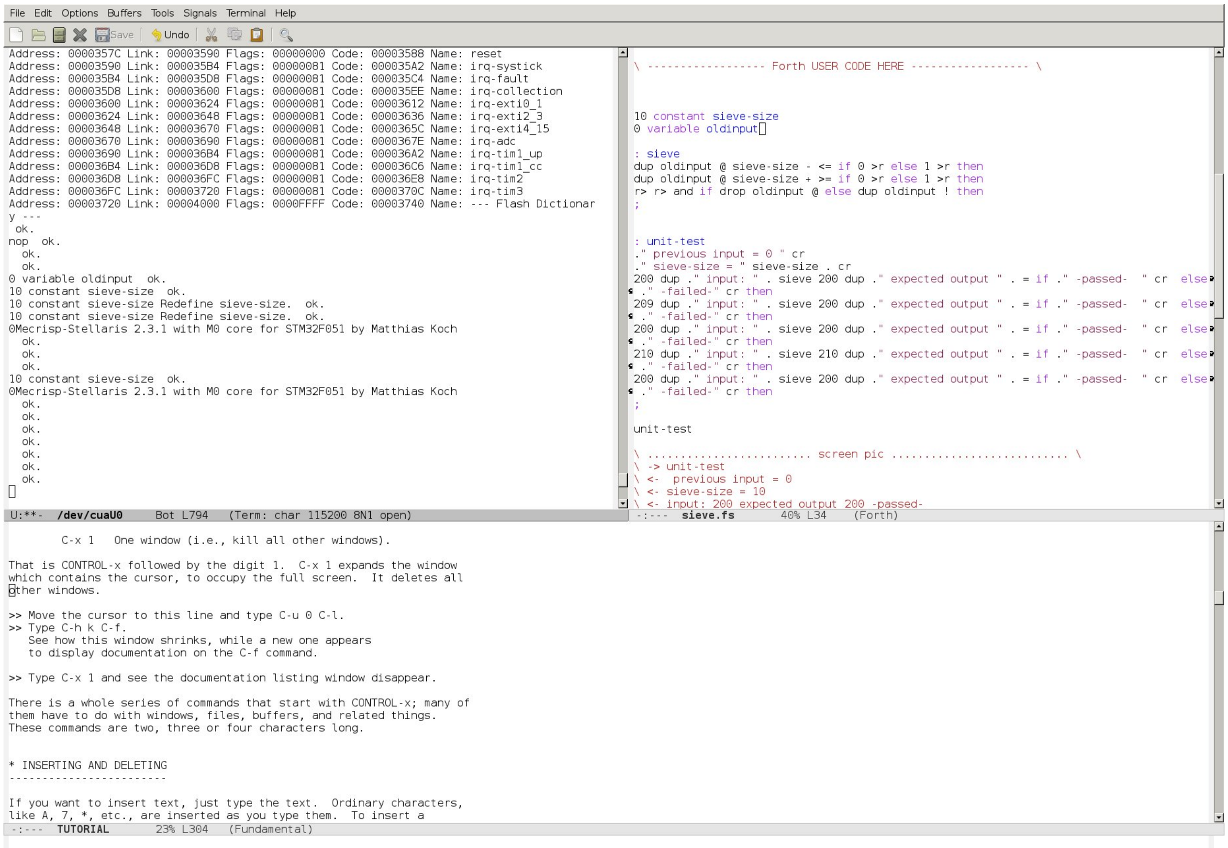
Split2 is 88 bytes long

```
see split2

200003B6: 2080  movs r0 #80
200003B8: 0500  lsls r0 r0 #14
200003BA: 30E0  adds r0 #E0
200003BC: 0080  lsls r0 r0 #2
200003BE: 6246  str r6 [ r0 #24 ]
200003C0: CF40  ldmia r7 { r6 }
200003C2: B500  push { lr }
200003C4: 2080  movs r0 #80
```

```
200003C6: 0500  lsls r0 r0 #14
200003C8: 30E0  adds r0 #E0
200003CA: 0080  lsls r0 r0 #2
200003CC: 6A43  ldr r3 [ r0 #24 ]
200003CE: 3F04  subs r7 #4
200003D0: 603E  str r6 [ r7 #0 ]
200003D2: 461E  mov r6 r3
200003D4: 2E00  cmp r6 #0
200003D6: D001  beq 200003DC
200003D8: 3F04  subs r7 #4
200003DA: 603E  str r6 [ r7 #0 ]
200003DC: 2E00  cmp r6 #0
200003DE: CF40  ldmia r7 { r6 }
200003E0: DD15  ble 2000040E
200003E2: 3F04  subs r7 #4
200003E4: 603E  str r6 [ r7 #0 ]
200003E6: 260A  movs r6 #A
200003E8: 2082  movs r0 #82
200003EA: 0180  lsls r0 r0 #6
200003EC: 3039  adds r0 #39
200003EE: 4780  blx r0  --> /mod
200003F0: 2080  movs r0 #80
200003F2: 0500  lsls r0 r0 #14
200003F4: 30E0  adds r0 #E0
200003F6: 0080  lsls r0 r0 #2
200003F8: 6246  str r6 [ r0 #24 ]
200003FA: CF40  ldmia r7 { r6 }
200003FC: 2086  movs r0 #86
200003FE: 01C0  lsls r0 r0 #7
20000400: 3021  adds r0 #21
20000402: 4780  blx r0  --> .
20000404: 2097  movs r0 #97
20000406: 0180  lsls r0 r0 #6
20000408: 3033  adds r0 #33
2000040A: 4780  blx r0  --> cr
2000040C: E7DA  b 200003C4
2000040E: BD00  pop { pc }
```

Emacs and Forth

This is another work in progress for me. Emacs has a ‘terminal’ mode which allows one to edit in one Emacs sub window, run code on a live Mecrisp-Stellaris MCU in another Emacs sub window and so on.



```
Address: 0000357C Link: 00003590 Flags: 00000000 Code: 00003588 Name: reset
Address: 00003590 Link: 000035B4 Flags: 00000081 Code: 000035A2 Name: irq-systick
Address: 000035B4 Link: 000035D8 Flags: 00000081 Code: 000035C4 Name: irq-fault
Address: 000035D8 Link: 00003600 Flags: 00000081 Code: 000035EE Name: irq-collection
Address: 00003600 Link: 00003624 Flags: 00000081 Code: 00003612 Name: irq-exti0_1
Address: 00003624 Link: 00003648 Flags: 00000081 Code: 00003636 Name: irq-exti2_3
Address: 00003648 Link: 00003670 Flags: 00000081 Code: 0000365C Name: irq-exti4_15
Address: 00003670 Link: 00003690 Flags: 00000081 Code: 0000367E Name: irq-adc
Address: 00003690 Link: 000036B4 Flags: 00000081 Code: 000036A2 Name: irq-tim1_up
Address: 000036B4 Link: 000036D8 Flags: 00000081 Code: 000036C6 Name: irq-tim1_cc
Address: 000036D8 Link: 000036FC Flags: 00000081 Code: 000036E8 Name: irq-tim2
Address: 000036FC Link: 00003720 Flags: 00000081 Code: 0000370C Name: irq-tim3
Address: 00003720 Link: 00004000 Flags: 0000FFFF Code: 00003740 Name: --- Flash Dictionar
y ---
 ok.
nop  ok.
  ok.
  ok.
0 variable oldinput  ok.
10 constant sieve-size  ok.
10 constant sieve-size Redefine sieve-size.  ok.
10 constant sieve-size Redefine sieve-size.  ok.
0Mecrisp-Stellaris 2.3.1 with M0 core for STM32F051 by Matthias Koch
  ok.
  ok.
  ok.
10 constant sieve-size  ok.
0Mecrisp-Stellaris 2.3.1 with M0 core for STM32F051 by Matthias Koch
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.

U:**-  /dev/cuaU0     Bot L794   (Term: char 115200 8N1 open)
```

```
\ ----------------- Forth USER CODE HERE ----------------- \



10 constant sieve-size
0 variable oldinput

: sieve
dup oldinput @ sieve-size - <= if 0 >r else 1 >r then
dup oldinput @ sieve-size + >= if 0 >r else 1 >r then
r> r> and if drop oldinput @ else dup oldinput ! then
;


: unit-test
." previous input = 0 " cr
." sieve-size = " sieve-size . cr
200 dup ." input: " . sieve 200 dup ." expected output " . = if ." -passed-  " cr  else
 ." -failed-" cr then
209 dup ." input: " . sieve 200 dup ." expected output " . = if ." -passed-  " cr  else
 ." -failed-" cr then
200 dup ." input: " . sieve 200 dup ." expected output " . = if ." -passed-  " cr  else
 ." -failed-" cr then
210 dup ." input: " . sieve 210 dup ." expected output " . = if ." -passed-  " cr  else
 ." -failed-" cr then
200 dup ." input: " . sieve 200 dup ." expected output " . = if ." -passed-  " cr  else
 ." -failed-" cr then
;

unit-test

\ ........................ screen pic .......................... \
\ -> unit-test
\ <-  previous input = 0
\ <- sieve-size = 10
\ <- input: 200 expected output 200 -passed-
-:---  sieve.fs       40% L34    (Forth)
```

```
        C-x 1   One window (i.e., kill all other windows).

That is CONTROL-x followed by the digit 1.  C-x 1 expands the window
which contains the cursor, to occupy the full screen.  It deletes all
other windows.

>> Move the cursor to this line and type C-u 0 C-l.
>> Type C-h k C-f.
   See how this window shrinks, while a new one appears
   to display documentation on the C-f command.

>> Type C-x 1 and see the documentation listing window disappear.

There is a whole series of commands that start with CONTROL-x; many of
them have to do with windows, files, buffers, and related things.
These commands are two, three or four characters long.


* INSERTING AND DELETING
------------------------

If you want to insert text, just type the text.  Ordinary characters,
like A, 7, *, etc., are inserted as you type them.  To insert a
-:---  TUTORIAL       23% L304   (Fundamental)
```

Emacs with a serial window talking to a stm32 target

To Do:

- either learn how to program LISP to insert serial line delays when uploading to the mecrisp-stellaris target

- or use serial handshaking in emacs and the target

Endianess

A interesting article from https://fgiesen.wordpress.com/2014/10/25/little-endian-vs-big-endian/

Little-endian vs. big-endian

October 25, 2014

Little-endian (LE) vs. big-endian (BE) and some of the trade-offs involved. The whole debate comes up periodically by proponents of LE or BE with missionary zeal, and then I get annoyed, because usually what makes either endianness superior for some applications makes it inferior for others. So for what it’s worth, here’s the trade-offs I’m aware of:

Doing math vs. indexing/sorting/searching

LE stores bytes in the order you do most math operations on them (if you were to do it byte by byte and not in larger chunks, that is). Additions and subtractions proceed from least-significant bit (LSB) to most-significant bit (MSB), always, because that’s the order carries (and borrows) are generated. Multiplications form partial products from smaller terms (at the limit, individual bits, though for hardware you’re more likely to use radix-4 booth recoding or similar) and add them, and the final addition likewise is LSB to MSB. Long division is the exception and works its way downwards from the most significant bits, but divisions are generally much less frequent than additions, subtractions and multiplication.

Arbitrary-precision arithmetic (“bignum arithmetic”) thus typically chops up numbers into segments (“legs”) matching the word size of the underlying machine, and stores these words in memory ordered from least significant to most significant – on both LE and BE architectures.

All 8-bit ISAs I’m personally familiar with (Intel 8080, Zilog Z80, MOS 6502) use LE, presumably for that reason; it’s the more natural byte order for 16-bit numbers if you only have an 8-bit ALU. (That said, Motorola’s 8-bit 6800 apparently used BE). And consequently, if you’re designing a new architecture with the explicit goal of being source-code compatible with the 8080 (yes, x86 was already constrained by backwards-compatibility considerations even for the original 8086!) it’s going to be little-endian.

BE stores bytes in the order you compare them (assuming a lexicographic compare).

So if you want to do a lexicographic sort, memcmp does the right thing on BE but not on LE: encoding numbers in BE is an order-preserving bijection (if the ordering predicate is lexicographic comparison). This is a very useful property if you’re in the business of selling your customers machines that spend a large chunk of their time sorting, searching and retrieving records, and likely one of the reasons why IBM’s architectures dating back as far as the mainframe era are big-endian. (It doesn’t make sense to speak of “endianness” before the IBM 7030, since that was the machine that introduced byte-addressable memory in the first place; machines before that point had word-based memory). It’s still common to encode numbers in BE for databases and key-value stores, even on LE architectures.

Byte order vs. bit order

All LE architectures I’m aware of have the LSB as bit 0 and number both bits and bytes in order of increasing significance. Thus, bit and byte order agree: byte 0 of a number on a LE machine stores bits 0-7, byte 1 stores bits 8-15 and so forth. (Assuming 8-bit bytes, that is.)

BE has two schools. First, there’s “Motorola style”, which is bits numbered with LSB=0 and from then on in increasing order of significance; but at the same time, byte 0 is the most significant byte, and following bytes decrease in significance. So by these conventions, a 16-bit number would store bits 8-15 in byte 0, and bits 0-7 in byte 1. As you can see, there’s a mismatch between byte order and bit order.

Finally, there’s BE “IBM style”, which instead labels the MSB as bit 0. As the bit number increases, they decrease in significance. In this scheme, same as in LE, byte 0 stores bits 0-7 of a number, byte 1 stores bits 8-15, and so forth; these bytes are exactly reversed compared to the LE variant, but bit and byte ordering are in agreement again.

That said, referring to the MSB as bit 0 is confusing in other ways; people normally expect bit 0 to have mask 1 << 0, and with MSB-first bit numbering that’s not the case.

Memory access

For LE, the 8/16/32-bit prefixes of a 64-bit number all start at the same address as the number itself. This can be viewed as either an advantage (“it’s convenient!”) or a disadvantage (“it hides bugs!”).

A LE load of 8/16/32/64 bits will always put all source bits at the same position in the destination register; as you make the load wider, it will just zero-clear (or sign-extend) less of them. Flow of data through the load/store circuitry is thus essentially the same regardless of operand size; different AND masks corresponding to the load size, but that’s it.

For BE, prefixes start at different offsets. Again, can be viewed as either an advantage (“it prevents bugs!”) or a disadvantage (“I can’t transparently widen fields after the fact!”).

A BE load of 8/16/32/64 bits puts the source bits in different locations in the destination register; instead of a width-dependent mask, we get a width-dependent shift. In a circuit, this is a Mux of differently-shifted versions of the source operand, which is (very slightly) more complicated than the masking for LE. (Not that I actually think anyone cares about HW complexity at that level today, or has in over a decade for that matter.)

That said, the difference can be slightly interesting if you don’t have a full complement of differently-sized loads; the Cell SPUs are an example. If you don’t have narrow loads, BE is hit a bit more than LE is. A synthesized LE narrow load is wide_unaligned_load(addr) & mask (where the wide unaligned load might itself consist of multiple steps, like it does on the SPUs); synthesized BE narrow load is wide_unaligned_load(addr + offs) & mask. Note the extra add of a non-zero offset, which means one more instruction. You can get rid of it in principle by just having all addresses for e.g. byte-aligned data be pre-incremented by offs, but that’s obnoxious too.

Example Forth Programs

How to check the Return Stack is balanced, and splitting a number into digits

Using the INIT word for turnkey applications

Examples: Word Usage

Brief Word examples, linked from the Dictionary page

WFI ($BF30)

WFI is a hint instruction that suspends execution until one of the following events occurs:

- An exception
- An interrupt becomes pending which would preempt if PRIMASK was clear
- A Debug Entry request, regardless of whether Debug is enabled.

WFI is intended for power saving only. When writing software assume that WFI might behave as a NOP operation.

```
: wfi ( -- mcu asleep ) [ $BF30 h, ] inline ;  \ WFI Opcode, Wait For Interrupt, enters sleep mode
```

```
see wfi
2000055E: B500  push { lr }
20000560: BF30
20000562: BD00  pop { pc }
ok.
```

Special WFI Insight Words from Matthias Koch

```
: wfi ( -- ) ." SRSLY ? Consider using MSP430 instead !" cr ;

: wfe ( -- ) ." Events ? Check your local newspaper for events ! Press any key to continue." cr key drop ;
```

<builds does>

This example prints some GPIO Modes, i.e. IN, OUT, ANALOG, etc to demonstrate the versatility of <builds does>, often described as the ‘jewel of forth’.

- Developed on: ST32F0 Discovery board
- MCU: Cortex M0

```
\ Peripheral constants
\ -------------------------------------------------------------
-------- \
   $48000000   constant GPIOA_MODER
   $48000400   constant GPIOB_MODER
   $48000800   constant GPIOC_MODER

\ Register inits, note: This is a mecrisp-stellaris machine
and
\ GPIOA-2 and 3 are already defined as AF and used as USART 1
\ -------------------------------------------------------------
-------- \
 %01  0 lshift GPIOA_MODER bis!        \ GPIOA_MODER0 as
output
 %11  8 lshift GPIOA_MODER bis!        \ GPIOA_MODER4 as
analog input

\ The <builds  does> example
\ -------------------------------------------------------------
-------- \
: moder? cr <builds , , does>  2@
   dup >r lshift and r> rshift
   case
   %00 of ." IN "     cr endof
   %01 of ." OUT "    cr endof
   %10 of ." AF "     cr endof
   %11 of ." ANALOG " cr endof
   endcase
 ;

\ Create moderx? words
\ -------------------------------------------------------------
-------- \
 %11  0 moder?  moder0?
 %11  2 moder?  moder1?
 %11  4 moder?  moder2?
 %11  6 moder?  moder3?
 %11  8 moder?  moder4?
```

```
\ Find the moderx? modes on a live system
\ ------------------------------------------------------------
-------- \
 gpioa_moder @ moder0?
 gpioa_moder @ moder1?
 gpioa_moder @ moder2?
 gpioa_moder @ moder3?
 gpioa_moder @ moder4?

\  Actual realtime status of gpioa_moder
\  gpioa_moder. GPIOA_MODER (read-write) $000003A1
\  15| 14| 13| 12| 11| 10| 09| 08| 07| 06| 05| 04| 03| 02| 01| 00
\  00 00 00 00 00 00 00 00 00 00 00 11 10 10 00 01

\   gpioa_moder @ moder0? OUT
\   gpioa_moder @ moder1? IN
\   gpioa_moder @ moder2? AF
\   gpioa_moder @ moder3? AF
\   gpioa_moder @ moder4? ANALOG

\ Previously defined moderx? words apply to gpiob and gpioc
also
\ ------------------------------------------------------------
-------- \
\   gpiob_moder @ moder0? IN
\   gpioc_moder @ moder0? IN
```

bit@

```
GPIOA_IDR (read-only) $0000B0F8
1| 1| 1| 1| 1| 1|
5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0
1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 0

: pa0?
  ." gpioa bit 0 "
  %1 0 lshift gpioa_idr bit@
     if ." is high" cr
     else ." is low" cr
  then ;

: pa3?
  ." gpioa bit 3 "
  %1 3 lshift gpioa_idr bit@
     if ." is high" cr
     else ." is low" cr
  then ;

pa0? gpioa bit 0 is low
ok.
pa3? gpioa bit 3 is high
ok.
```

compare

```
2dup s" tos" compare if 2drop 6 exit then
2dup s" psp" compare if 2drop 7 exit then
2dup s" sp"  compare if 2drop 13 exit then
2dup s" lr"  compare if 2drop 14 exit then
2dup s" pc"  compare if 2drop 15 exit then
```

+loop

```
: backwards ( -- )
  0 10 do I . -1 +loop ;

backwards 10 9 8 7 6 5 4 3 2 1 0  ok.
```

+LOOP can be misused to set I to any value e.g. to set it to 13 if has the value 5

```
: misused cr 20 0 do i . i 5 = if 8 else 1 then +loop ;

misused
0 1 2 3 4 5 13 14 15 16 17 18 19  ok.
```

loop

```
: forwards ( -- )
  10 0 do I . loop ;

forwards 0 1 2 3 4 5 6 7 8 9  ok.
```

min

```
10 20 min . 10  ok.
```

max

```
10 20 max . 20  ok.
```

min-max window

```
100 0 max 63 min . 63  ok.

-10 0 max 63 min . 0  ok.
```

umin

```
-10 0 umax . -10  ok.
```

Return Stack

Return Stack Words

```
>r
r@
r>
rdrop
rdepth
rpick
2>r
2r@
2r>
2rdrop
```

Warning

It’s often convenient to be able to put a stack item temporarily in the return stack, but the following rules must be observed.

- Data put on the return stack must be taken back within the same word.
- Data put on the return stack outside a ?DO (DO) … LOOP (+LOOP) cannot be accessed within the loop.
- Data put on the return stack within a ?DO (DO) … LOOP (+LOOP) must be taken back before leaving the loop.
- Data cannot be on the return stack when executing I or J in a loop.

A multasking delay

```
: msdelay ( u -- ) cr        \ delay for x milliseconds
 >r                          \ save delay value on the return
stack
 0 ms_second_counter !       \ reset ms_second_counter to
zero. "ms_second_counter" is incremented by Systick every 1
ms
 begin
 r@                          \ get the delay value from the
return stack
 ms_second_counter @         \ get ms_second_counter value
 =                           \ loop until equal to
ms_second_counter
 pause                       \ allows a cooperative
multitasker to use the dead time between each millisecond
 until
 r> drop                     \ balance the return stack
 ;                           \ finished, delay is over
```

Roll, -Roll

```
1 2 3 4 5  ok.
4 roll .s [5 ] 42 2 3 4 5 1  ok.
4 roll .s [5 ] 42 3 4 5 1 2  ok.
4 roll .s [5 ] 42 4 5 1 2 3  ok.
4 roll .s [5 ] 42 5 1 2 3 4  ok.
4 roll .s [5 ] 42 1 2 3 4 5  ok.
4 -roll .s [5 ] 42 5 1 2 3 4  ok.
4 -roll .s [5 ] 42 4 5 1 2 3  ok.
```

```
4 -roll .s [5 ] 42 3 4 5 1 2  ok.
4 -roll .s [5 ] 42 2 3 4 5 1  ok.
4 -roll .s [5 ] 42 1 2 3 4 5  ok.
```

File System

> The block file system is the traditional forth file system. It has the major advantage that it’s very simple to implement on any hardware. The logic and beauty of forth is that by the time you have read the datasheet for a device you have it working… if the datasheet is accurate. The traditional forth file is a block of sixteen 64 character lines which are referenced by number. Forth was designed to work on a system with very little RAM and a small disk drive so you had words which compiled certain blocks into RAM when you wanted to use that program. Then you executed forget and loaded the next program from block memory using a word that knew which blocks that program was stored in. Many MCUs today are several orders of magnitude more powerful than the machines forth originated on. [reginald.beardsley]

Mecrisp-Stellaris Block Implementation

`Download:blocks.fs`

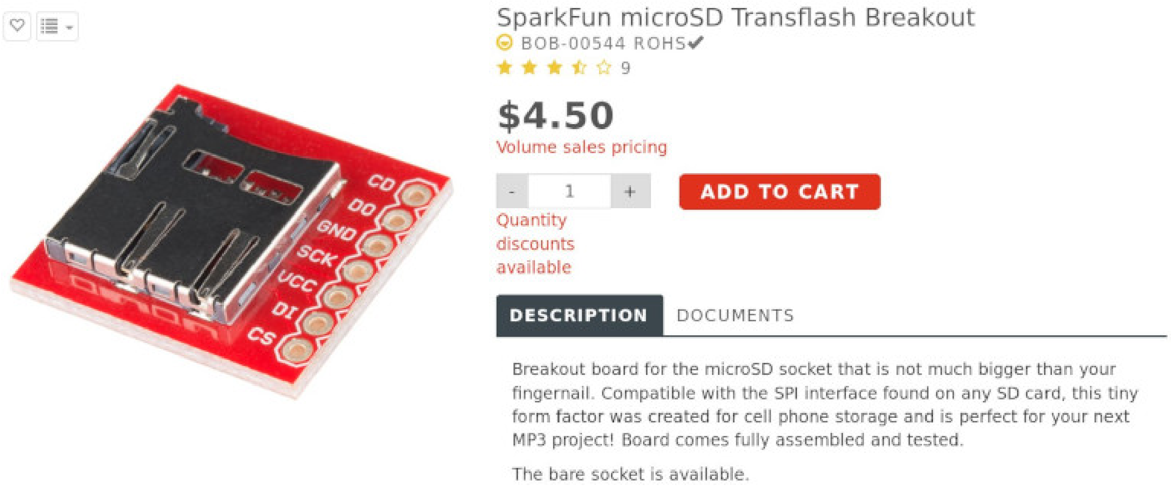
Block Storage Hardware

Note

In 1987 I added a 3.5” floppy disk to a Embedded Forth system running a Rockwell 65F11 Forth chip. I didn’t write the code, I only built the hardware which used a floppy controller chip and a generic floppy drive. It worked fine, but these days one would use a SD Card.

3.5” 1.2MB Floppy Drive, circa 1987.

SDcard



SparkFun microSD Transflash Breakout

BOB-00544 ROHS✔

★★★☆☆ 9

$4.50

Volume sales pricing

Quantity discounts available

Breakout board for the microSD socket that is not much bigger than your fingernail. Compatible with the SPI interface found on any SD card, this tiny form factor was created for cell phone storage and is perfect for your next MP3 project! Board comes fully assembled and tested.

The bare socket is available.

SD Card Links

Mbed SD Card documentation

https://os.mbed.com/cookbook/SD-Card-File-System

How to Use MMC/SDC

http://www.elm-chan.org/docs/mmc/mmc_e.html

Forth Block Links

Forth Block Device Primer

http://www.shaels.net/index.php/mv4th-forth/mv4thdocumentscat/134-using-forth-block-device-to-execute-dynamic-code

Gforth Block Documentation

https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Blocks.html 5.18 Blocks [gforth]

When you run Gforth on a modern desk-top computer, it runs under the control of an operating system which provides certain services. One of these services is file services, which allows Forth source code and data to be stored in files and read into Gforth (see Files).

Traditionally, Forth has been an important programming language on systems where it has interfaced directly to the underlying hardware with no intervening operating system. Forth provides a mechanism, called blocks, for accessing mass storage on such systems.

A block is a 1024-byte data area, which can be used to hold data or Forth source code. No structure is imposed on the contents of the block. A block is identified by its number; blocks are numbered contiguously from 1 to an implementation-defined maximum.

A typical system that used blocks but no operating system might use a single floppy-disk drive for mass storage, with the disks formatted to provide 256-byte sectors. Blocks would be implemented by assigning the first four sectors of the disk to block 1, the second four sectors to block 2 and so on, up to the limit of the capacity of the disk. The disk would not contain any file system information, just the set of blocks.

On systems that do provide file services, blocks are typically implemented by storing a sequence of blocks within a single blocks file. The size of the blocks file will be an exact multiple of 1024 bytes, corresponding to the number of blocks it contains. This is the mechanism that Gforth uses.

Only one blocks file can be open at a time. If you use block words without having specified a blocks file, Gforth defaults to the blocks file blocks.fb. Gforth uses the Forth search path when attempting to locate a blocks file (see Source Search Paths).

When you read and write blocks under program control, Gforth uses a number of block buffers as intermediate storage. These buffers are not used when you use load to interpret the contents of a block.

The behaviour of the block buffers is analagous to that of a cache. Each block buffer has three states:

```
Unassigned
Assigned-clean
Assigned-dirty
```

Initially, all block buffers are unassigned. In order to access a block, the block (specified by its block number) must be assigned to a block buffer.

The assignment of a block to a block buffer is performed by block or buffer. Use block when you wish to modify the existing contents of a block. Use buffer when you don’t care about the existing contents of the block1.

Once a block has been assigned to a block buffer using block or buffer, that block buffer becomes the current block buffer. Data may only be manipulated (read or written) within the current block buffer.

When the contents of the current block buffer has been modified it is necessary, before calling block or buffer again, to either abandon the changes (by doing nothing) or mark the block as changed (assigned-dirty), using update. Using update does not change the blocks file; it simply changes a block buffer’s state to assigned-dirty. The block will be written implicitly when it’s buffer is needed for another block, or explicitly by flush or save-buffers.

word Flush writes all assigned-dirty blocks back to the blocks file on disk. Leaving Gforth with bye also performs a flush.

In Gforth, block and buffer use a direct-mapped algorithm to assign a block buffer to a block. That means that any particular block can only be assigned to one specific block buffer, called (for the particular operation) the victim buffer. If the victim buffer is unassigned or assigned-clean it is allocated to the new block immediately. If it is assigned-dirty its current contents are written back to the blocks file on disk before it is allocated to the new block.

Although no structure is imposed on the contents of a block, it is traditional to display the contents as 16 lines each of 64 characters. A block provides a single, continuous stream of input (for example, it acts as a single parse area) – there are no end-of-line characters within a block, and no end-of-file character at the end of a block. There are two consequences of this:

> The last character of one line wraps straight into the first character of the following line The word – comment to end of line – requires special treatment; in the context of a block it causes all characters until the end of the current 64-character “line” to be ignored.

In Gforth, when you use block with a non-existent block number, the current blocks file will be extended to the appropriate size and the block buffer will be initialised with spaces.

Gforth includes a simple block editor (type use blocked.fb 0 list for details) but doesn’t encourage the use of blocks; the mechanism is only provided for backward compatibility – ANS Forth requires blocks to be available when files are.

Common techniques that are used when working with blocks include:

> A screen editor that allows you to edit blocks without leaving the Forth environment. Shadow screens; where every code block has an associated block containing comments (for example: code in odd block numbers, comments in even block numbers). Typically, the block editor provides a convenient mechanism to toggle between code and comments. Load blocks; a single block (typically block 1) contains a number of thru commands which load the whole of the application.

See Frank Sergeant’s Pygmy Forth to see just how well blocks can be integrated into a Forth programming environment.

`open-blocks       c-addr u –         gforth       “open-blocks”`

Use the file, whose name is given by c-addr u, as the blocks file.

`use       "file" –         gforth       “use”`

Use file as the blocks file.

`block-offset       – addr         gforth       “block-offset”`

User variable containing the number of the first block (default since 0.5.0: 0). Block files created with Gforth versions before 0.5.0 have the offset 1. If you use these files you can: 1 offset !; or add 1 to every block number used; or prepend 1024 characters to the file.

`get-block-fid       – wfileid         gforth       “get-block-fid”`

Return the file-id of the current blocks file. If no blocks file has been opened, use blocks.fb as the default blocks file.

`block-position       u –         block       “block-position”`

Position the block file to the start of block u.

`list       u –         block-ext       “list”`

Display block u. In Gforth, the block is displayed as 16 numbered lines, each of 64 characters.

`scr       – a-addr         block-ext       “s-c-r”`

User variable – a-addr is the address of a cell containing the block number of the block most recently processed by list.

`block       u – a-addr         block       “block”`

If a block buffer is assigned for block u, return its start address, a-addr. Otherwise, assign a block buffer for block u (if the assigned block buffer has been updated, transfer the contents to mass storage), read the block into the block buffer and return its start address, a-addr.

`buffer       u – a-addr         block       “buffer”`

If a block buffer is assigned for block u, return its start address, a-addr. Otherwise, assign a block buffer for block u (if the assigned block buffer has been updated, transfer the contents to mass storage) and return its start address, a-addr. The subtle difference between buffer and block mean that you should only use buffer if you don’t care about the previous contents of block u. In Gforth, this simply calls block.

`empty-buffers       –         block-ext       “empty-buffers”`

Mark all block buffers as unassigned; if any had been marked as assigned-dirty (by update), the changes to those blocks will be lost.

`empty-buffer       buffer –         gforth       “empty-buffer”`

`update       –         block       “update”`

Mark the state of the current block buffer as assigned-dirty.

`updated?       n – f         gforth       “updated?”`

Return true if updated has been used to mark block n as assigned-dirty.

`save-buffers – block “save-buffers”`

Transfer the contents of each updated block buffer to mass storage, then mark all block buffers as assigned-clean.

`save-buffer buffer – gforth “save-buffer”`

`flush – block “flush”`

Perform the functions of save-buffers then empty-buffers.

`load i*x n – j*x block “load”`

Save the current input source specification. Store n in BLK, set >IN to 0 and interpret. When the parse area is exhausted, restore the input source specification.

`thru i*x n1 n2 – j*x block-ext “thru”`

load the blocks n1 through n2 in sequence.

`+load i*x n – j*x gforth “+load”`

Used within a block to load the block specified as the current block + n.

`+thru i*x n1 n2 – j*x gforth “+thru”`

Used within a block to load the range of blocks specified as the current block + n1 thru the current block + n2.

`--> – gforth “chain”`

If this symbol is encountered whilst loading block n, discard the remainder of the block and load block n+1. Used for chaining multiple blocks together as a single loadable unit. Not recommended, because it destroys the independence of loading. Use thru (which is standard) or +thru instead.

`block-included a-addr u – gforth “block-included”`

Use within a block that is to be processed by load. Save the current blocks file specification, open the blocks file specified by a-addr u and load block 1 from that file (which may in turn chain or load other blocks). Finally, close the blocks file and restore the original blocks file.

F0 Discovery: 48 MHz Clock

- STM32F0 Discovery Board using a STM32F0xx MCU
- Utilises the 8MHz MCO crystal controlled frequency from the SWD programmer
- Increases the default Mecrisp-Stellaris speed by 6 times to the rated maximum of 48 MHz
- Serial comms runs at 460800 Baud. For 115200 Baud change USART1_BRR to 208
- Power usage STM32F051: 12mA (3mA @ 8MHz)
- Maximum rated MCU speed
- standalone, no other files needed

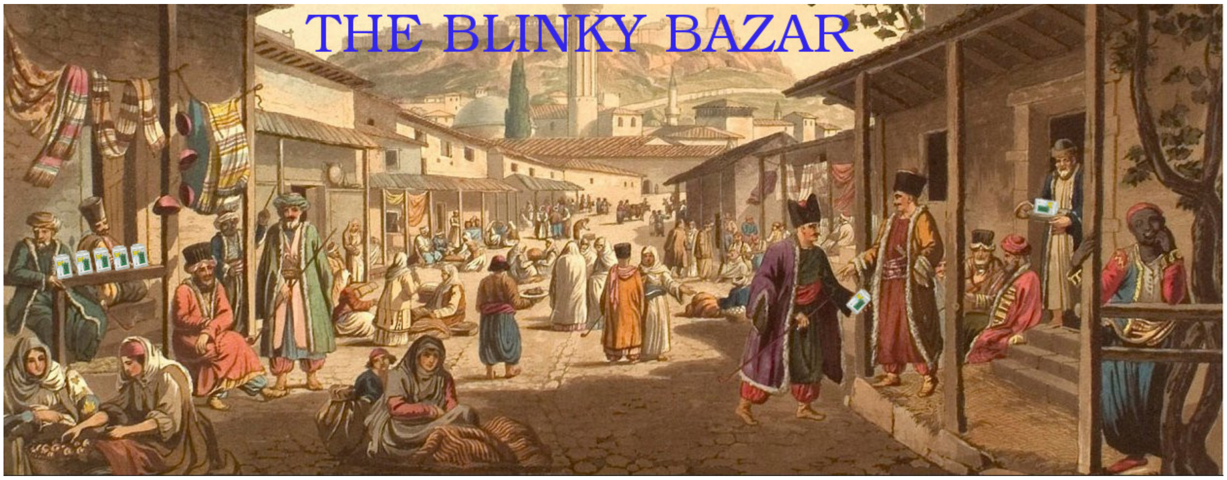
Download: `f0-disco-48mhz.fs`

Sourcecode

```
\ Program Name: f0-disco-48mhz.fs
\ This program is standalone, no other files are needed
\ Date: Sat Apr 28 08:56:40 AEST 2018
\ Copyright 2018  t.porter <terry@tjporter.com.au>, licensed under the GPL
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F051, Board: STM32F0 Discovery Board
\ All register names are CMSIS-SVD compliant
\ This Program : Increases the System Clock to 48 MHz using the Xtal controlled oscillator from the
\ F0 Discovery Board SWD subsystem. Can be easily be changed to HSI, see RCC_CFGR_PLLSRC below
\ ---------------------------------------------------------------------------\

\ compiletoflash
compiletoram
```

```
 $4001380C constant USART1_BRR ( Baud rate register )

: FLASH_ACR_LATENCY ( %XXX -- )  0  lshift $40022000 ;  (
LATENCY )
: FLASH_ACR_PRFTBE %1  4  lshift $40022000 ;            (
PRFTBE Prefetch buffer enable )
: RCC_CR_HSEON %1  16  lshift $40021000 ;               (
External High Speed clock  enable )
: RCC_CR_HSERDY? %1  17  lshift $40021000 bit@ ;        (
External High Speed clock ready  flag )
: RCC_CFGR_PLLMUL ( %XXXX -- )  18  lshift $40021004 ;  ( PLL
Multiplication Factor )
: RCC_CFGR_PLLSRC ( %XX -- )  15  lshift $40021004 ;    ( PLL
clock source: %10: HSE/PREDIV, 01: HSI/PREDIV  )
: RCC_CR_PLLON %1  24  lshift $40021000 ;               ( PLL
enable )
: RCC_CR_PLLRDY? %1  25  lshift $40021000 bit@ ;        ( PLL
clock ready flag )
: RCC_CFGR_SW ( %XX -- )  0  lshift $40021004 ;         (
System clock Switch: $10: PLL selected as system clock)
: RCC_CFGR_PPRE ( %XXX -- )  8  lshift $40021004  ;     ( APB
Low speed prescaler  APB1 )
: RCC_CFGR_HPRE ( %XXXX -- )  4  lshift $40021004 ;     ( AHB
prescaler )

: f0-disco-48mhz
\ Set flash wait states to 1, turn on prefetch bit. One wait
state, if 24 MHz < SYSCLK  48 MHz
%001  FLASH_ACR_LATENCY bis! \ LATENCY --> max = 1 for the
stm32f0, note more latency uses less power
FLASH_ACR_PRFTBE bis!

\ Enable HSE oscillator
RCC_CR_HSEON bis!
BEGIN  RCC_CR_HSERDY? UNTIL  \ Is HSE ready ?

\ Set PLL multiplication factor to 6, clock is 8Mhz system
clock
%0100 RCC_CFGR_PLLMUL bis!   \ %0100: PLL input clock x 6
%10 RCC_CFGR_PLLSRC bis!     \ %10: HSE/PREDIV, 01:
HSI/PREDIV

\ Enable PLL
RCC_CR_PLLON bis!
```

```
BEGIN RCC_CR_PLLRDY? UNTIL   \ Is PLL ready ?

\ Switch to PLL, set ABP and AHB Prescalers
%10 RCC_CFGR_SW bis!         \ System clock Switch: %10 PLL
selected as system clock
%100  RCC_CFGR_PPRE bis!     \ APB Low speed prescaler APB1
(pclock): %100: HCLK divided by 2
%0000 RCC_CFGR_HPRE bis!     \ AHB prescaler: 0xxx: SYSCLK
not divided

\ Update the Baud Rate Register so the serial comms is still
460800 Baud.
\ ((pll_multiplier * clock)/Baud rate)/2 = ((6*
8000000)/460800)/2 = 52
52 USART1_BRR !  \ for 115200 Baud change to 208
;

f0-disco-48mhz

compiletoram
```



F0 Discovery Blinkies

A dubious and probably fake rumour claims that Bill Gates once said *‘640 Blinkies should be enough for anyone’* and I have to agree as I have only managed to make 8 Blinkies so far. Six are **Forth** powered and the seventh is written in **Assembly language** and just 28 bytes long. I’ve included a eighth ‘C’ Language blinky for completness.

Please step into the Blinky Bazar, we have Blinky programs to suit your STM32F0 Discovery Board, just help yourself!

What *is* a blinky ?

In the *programming world*, a beginners first program usually prints ‘Hello World’.

In the *embedded world* our first **Forth** program usually blinks a LED, and we call this a *‘Blinky’*.

This page is a collection of *Blinkies* that run on the STM32 Discovery Board (or any STM32F Cortex M0 running Mecrisp-Stellaris Forth with LED's on GPIO's PC-8 and PC-9).

For the technical reference to the MCU, see here.

Starting with simple Blinkies and proceeding to the more complex and useful ones, this article was created in the hope of demonstrating some of the versatility of the Forth Programming Language.

I don't go into much detail about Forth or the STM32F0 MCU as these are *massive* areas, and the study of each one alone is worthy of considerable amounts of your time, but this is as good a place as any to start, so let's get going ...

Note

Wondering how to install Mecrisp-Stellaris Forth ?

Note

If entering any programs into the terminal by hand, don't enter lines beginning with *blackslash* as they're only comments for humans and Forth ignores them anyway.

Blinky 1

The simplest Blinky of all

`blinky-1.fs`

- Requires no other files, Just upload it, or type the code by hand straight into your Forth Terminal.
- All it does is blink, … forever.
- Disadvantage: This Blinky takes up 100% of the MCU time, it won’t even answer the terminal, which will be frozen.

Blinky 2

Introducing Register Names and Memory Mapping

`blinky-2.fs`

- Requires no other files, Just upload it, (or enter it by hand)
- Introduces the concept of Register Names and Memory Mapping, to simplify Source File content.
- To find out more about STM32 Register names, what they do etc, please refer to the STM32F Cortex F0 manual for more information.
- GPIOC_MODER selects the GPIOC MODES such as input, output, analog etc. In this case, it’s Output Mode.
- GPIOC_BSRR sets or resets GPIOC BIT or BITs, in this clase BIT 9 is being SET
- GPIOC_BRR resets (clears) GPIOC BIT or BITs, in this clase BIT 9 is being RESET
- Blinky 2 is otherwise identical in operation to Blinky 1.

Blinky 3

Introducing #require statements to preload vital library files

`blinky-3.fs` **`blinky-3-register-memory-map.fs`**

Note

This example introduces the very *important* concept of *preloading Forth files* via #require statements. This method will be used in the following Blinkies which are more complex and may have many more Memory Mapped Registers.

- One excellent Serial Terminal which supports #require statements is the e4thcom Serial Terminal.
- Why go to this trouble for a list of only four Register I hear you ask ?
- What if that list was actually 500 lines long or even 5000 lines ? If it was, then one would have to scroll down pages and pages of lists of Register Memory Mappings just to get to the actual program code.
- Blinky 3 is otherwise identical in operation to Blinky 1 and Blinky 2.

Blinky 4

Cooperative Multitasking.

`blinky-4.fs` **`blinky-3-register-memory-map.fs`** **`multitask.fs`**

- Blinky 4 uses Cooperative Multitasking, which means it blinks the LED *and* talks to you on the terminal at the same time ! Ok, it’s not *exactly* the same time, but it’s so fast, you won’t notice.
- Blinky 4 will also use the blinky-3-register-memory-map.fs.
- The same code to blink the LED is used as in Blinkies 1, 2, and 3, but this time it’s controlled by a Cooperative Multitasker. Try it, enter “reset” and the board will be

reset, and the LED will stop blinking (of course you'll have to load the program again, as *reset* clears the memory.
- Advantages: It's easy to code a number of separate tasks all doing their own thing and the terminal is fully active while the LED blinks.
- Disadvantages: Someone once said that *cooperative multitasking is like a group of kids passing a ball to each other, it works fine until one of the kids decides to keep the ball all to himself*.

Blinky 5

Interrupt Driven

`blinky-5.fs` **`blinky-5-register-memory-map.fs`**

- Blinky 5 blinks the LED exactly once per second, controlled by the MCU Systick Interrupt.
- Accurate and simple, the MCU isn't spending most of its time counting loops like Blinkies, 1,2,3, and 4
- Minimal MCU time is used, one interrupt means one change of state for the LED, leaving lots of time for other jobs.
- The terminal is fully active while the LED blinks

Note

The System Interrupt is already enabled by Mecrisp-Stellaris.

Blinky 6

Cooperative Multitasking *AND* Interrupts

`blinky-6.fs` **`blinky-5-register-memory-map.fs`** **`multitask.fs`**

- Blinky 5 blinks the LED exactly once per second, controlled by the MCU Systick Interrupt.
- Accurate and fairly simple, the MCU isn’t spending most of its time counting loops like Blinkies, 1,2,3, and 4
- Minimal MCU time is used, one interrupt means one change of state for the LED, leaving lots of time for other jobs.
- The terminal is fully active while the LED blinks
- Advantages: It’s easy to code a number of separate tasks all doing their own thing.
- Disadvantages: Someone once said that *cooperative multitasking is like a group of kids passing a ball to each other, it works fine until one of the kids decides to keep the ball all to himself*.

Blinky 7

A pure Assembly language Blinky. It’s a totally minimal, **28 Byte** standalone Blinky binary.

Written by Matthias Koch, Senior Assembly Wizzard, the creator of Mecrisp-Stellaris, it runs on a STM32F0 Discovery Board flashing the blue LED.

This is as small as we can make a Blinky binary for STM32F0, perhaps you can desgn a smaller one ? …. let me know and I’ll publish it here.

Download **`Tinyblinky Tarball Here`**. It has everything you need including a binary image ready to load.

- All it will ever do is blink the *selected* LED. Default is *blue*, but the green led can be used, it’s a option in the tinyblinky.s source file.

- The program uses a total of 28 bytes of Flash, which is way less than the minimum 19K bytes for Mecrisp-Stellaris alone.
- You can’t talk to it over a serial terminal, it’s not interactive like Mecrisp-Stellaris.
- The rest of the 64K Bytes of Flash in the MCU is wasted.

Blinky 8

Written in C, **`DOWNLOAD: stm32f0-C-blinky.zip`** complete with a Makefile, based on a helpful article by Frank Duignan.

Exercises For the Reader

To test your Forth Blinky skills …

1. Change the blinking LED COLOR ?
2. Change the rate of the blink ?
3. Add a second task so that both LED’s blink at different intervals
4. Add a third task, so that both LED’s blink at different intervals and the terminal prints “BEEP” (it’s a virtual beeper)

STM32F1 USB

A special **USB** Blue Pill Image that can be used to test if the USB on your board is working. It can also read the MCU internal ‘Device Electronic Signature’. Runs in a 64KB Flash STM32F103.

This uses the Mecrisp-Stellaris USB driver for STM32F103 by Jean-Claude Wippler which is based on the Coreforth USB driver by Eckhart Köppen.

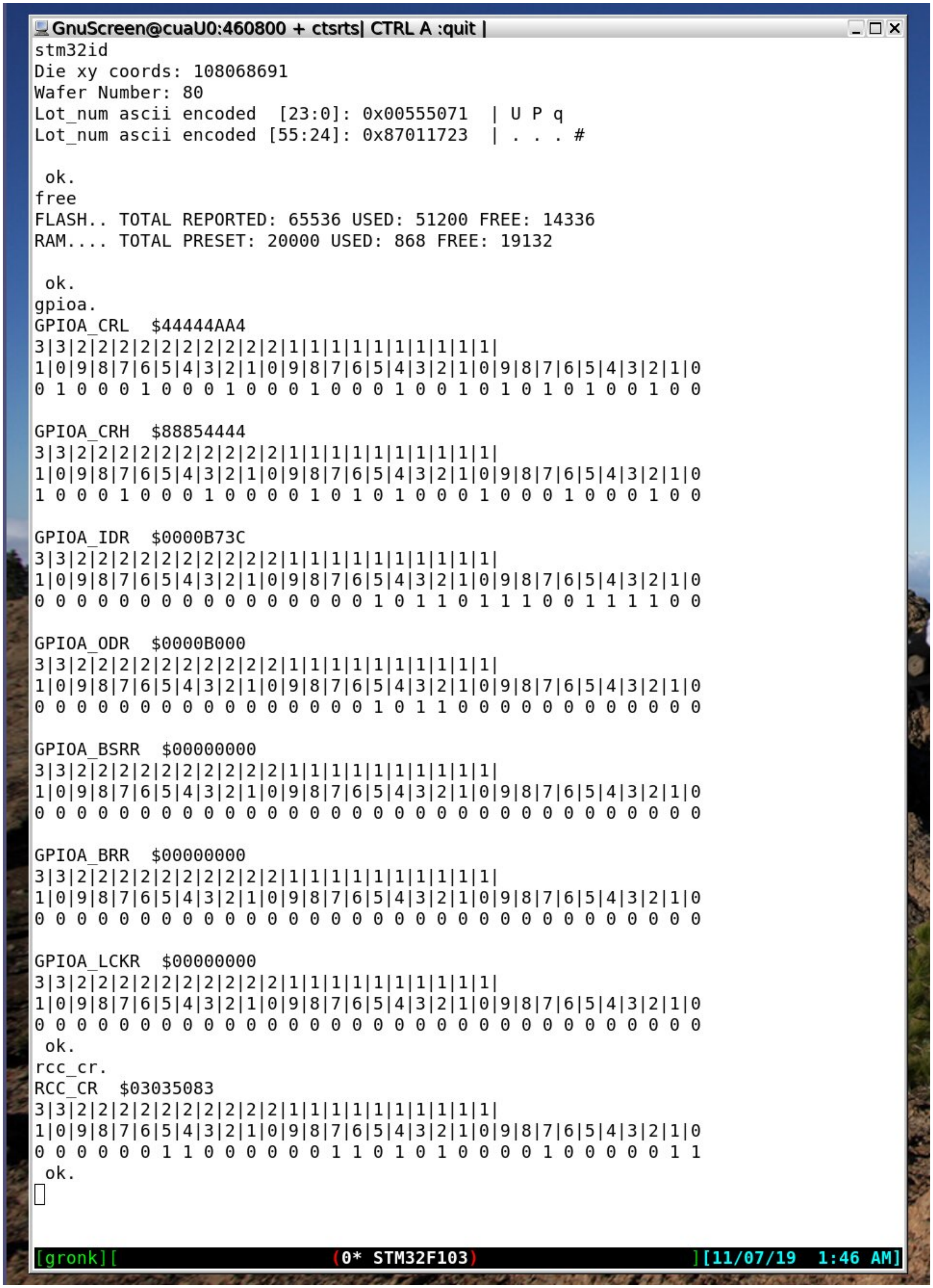
Note

From Viktor Dvořák. USB needs PC side idVendor = 0x0483 idProduct = 0x7540 driver which is STM32 Virtual COM port driver desribed in en.DM00478229.pdf. The driver is en.stsw-stm32102.zip (24755 kB) is available directly from st.com

Use it to check for *FAKE STM32F103 MCU’s* and let me know if it is useful ?

Note

if you don’t have a Blue Pill, read my rant before thinking about buying one ?



```
stm32id
Die xy coords: 108068691
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00555071  | U P q
Lot_num ascii encoded [55:24]: 0x87011723  | . . . #

 ok.
free
FLASH.. TOTAL REPORTED: 65536 USED: 51200 FREE: 14336
RAM.... TOTAL PRESET: 20000 USED: 868 FREE: 19132

 ok.
gpioa.
GPIOA_CRL  $44444AA4
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0

GPIOA_CRH  $88854444
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0

GPIOA_IDR  $0000B73C
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 0

GPIOA_ODR  $0000B000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0

GPIOA_BSRR  $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

GPIOA_BRR  $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

GPIOA_LCKR  $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 ok.
rcc_cr.
RCC_CR  $03035083
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 1
 ok.


[gronk][                      (0* STM32F103)                      ][11/07/19  1:46 AM]
```

```
Project Name: STM32F103 Created 20191107-0028 by terry porter
1882bdaff62f637a7bdd517d14f4c1a4.README: Specific Notes
regarding Mecrisp-Stellaris
Kernel 1882bdaff62f637a7bdd517d14f4c1a4.bin
-------------------------------------------------------------
-------------------------
Special 64 KB USB image. This image runs from the USB port on
any STM32F103.

1. Flash 1882bdaff62f637a7bdd517d14f4c1a4.bin image to board
using your usual method
2. Connect blue pill USB port to PC and look for  "<Mecrisp
STM32F10x Forth Serial Port>" device in your system
3. Connect a PC serial terminal to the USB device above, you
can choose any Baud Rate. NOTE: There is NO HANDSHAKING,
so a End Of Line Delay of 200ms will required for uploads.
4. You should see "MS RA 2.5.1 tpmod for STM32F103 by
Matthias Koch" at board power up

What special facilities exist on the board apart from all the
standard Mecrisp-Stellaris Words ?

1. "stm32id" for interrogating the electronic chip ID as
below. Perhaps it may find FAKES? let me know if it does ?
stm32id
Die xy coords: 108068691
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00555071  |  U P q
Lot_num ascii encoded [55:24]: 0x87011723  |  . . . #

2. "free"
FLASH.. TOTAL REPORTED: 65536 USED: 51200 FREE: 14336
RAM.... TOTAL PRESET: 20000 USED: 868 FREE: 19132

3. "words4"
Four column Word lister

4. "Dump" and "dump16"
gpioa hex. 40010800  ok.
$40010800 dump16 40010800 :  A4 4A 44 44 44 44 85 88   3C B7
00 00 00 B0 00 00  |  .JDDDD..  <....... |
 ok.

5. "bin."
1000 bin.
```

```
3322222222221111111111
10987654321098765432109876543210
00000000000000000000001111101000

6. all the registers for the following peripherals. Not all
peripherals could be included due to flash space
requirements.
PWR
RCC
GPIOA
GPIOB
GPIOC
AFIO
EXTI
DMA1
SDIO
RTC
TIM1
I2C1
SPI1
USART1
ADC1
NVIC

Example:
rcc_cfgr.
RCC_CFGR  $001D840A
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1
0
 ok.

gpioa.
GPIOA_CRL  $44444AA4
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0
0

GPIOA_CRH  $88854444
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
```

```
| 0
1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0

GPIOA_IDR  $0000B73C
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0
0

GPIOA_ODR  $0000B000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_BSRR  $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_BRR  $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_LCKR  $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 ok.
```

Download

Image tarball here: https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/mecrisp-stellaris-1882bdaff62f637a7bdd517d14f4c1a4.tar.gz/download

Note

all software on this site is released under the GPL V2 or BSD V3 licenses.

EEVblog Forum Feedback

https://www.eevblog.com/forum/microcontrollers/30kb-blue-pill-usb-binary-testing-image/

STM32ID

This program is based on the STM32F0 technical documentation because the STM32F1xx technical doc doesn’t give the format of the actual “device electronic Unique ID register” and many assume that the format is the same as the STM32F0.

I’m leaning towards the theory that they’re slightly different by way of the “lot numbers” because the none of the ASCII data for the F1 chips looks sensible to me so far. Perhaps they are “lot numbers” but just sequential decimal numbers without character based information ?

The “die” and ‘wafer number” data seems reasonable in all cases.

However in the F051 signature below, the ASCII lot number layout seems reasonable ? I’ll add to this when I get another F051 board set up to read.

NON STM32F103 MCU’s For Comparison

STM32F051

```
Die xy coords: 2752589
Wafer Number: 14
Lot_num ascii encoded  [23:0]: 0x00413657  |  A 6 W
Lot_num ascii encoded [55:24]: 0x20323336  |    2 3 6
```

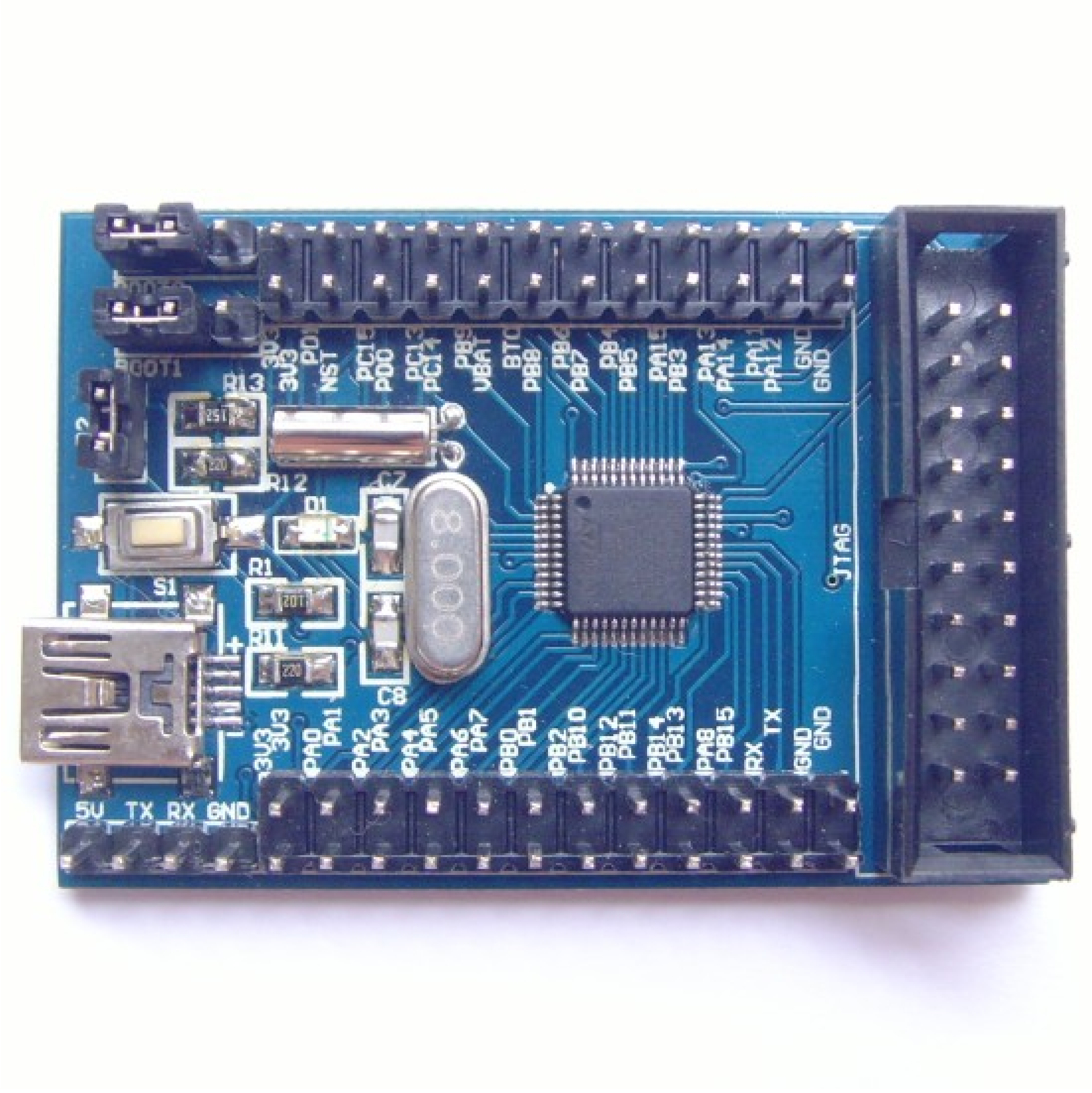
STM32L073

```
Die xy coords: 21444402
Wafer Number: 48
Lot_num ascii encoded  [23:0]: 0x00343837  |  4 8 7
Lot_num ascii encoded [55:24]: 0x00000011  |  . . . .
```

STM32F103 or Clone Results

1882bdaff62f637a7bdd517d14f4c1a4.bin User Results

This is the chip/board I developed my 1882bdaff62f637a7bdd517d14f4c1a4.bin image on. It’s in a $9 Shenzhen LC Technology board with 64KB Flash.



I bought 12 of these boards around 2014, one was dead out of the box, no option jumpers etc and one died while being flashed.

STM32F103C8T6 #1

```
Die xy coords: 108068691
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00555071  |  U P q
Lot_num ascii encoded [55:24]: 0x87011723  |  . . . #
```

STM32F103C8T6 #2

```
Die xy coords: 108068688
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00555071  |  U P q
Lot_num ascii encoded [55:24]: 0x87083408  |  . . 4 .
```

Note

#1 is very close to this one, same silicon wafer only 3 positions away!

STM32F103C8T6 #3

```
Die xy coords: 108003144
Wafer Number: 73
Lot_num ascii encoded  [23:0]: 0x00515389  |  Q S .
Lot_num ascii encoded [55:24]: 0x87084441  |  . . D A
```

STM32F103C8T6 #4

```
Die xy coords: 107675464
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00485366  |  H S f
Lot_num ascii encoded [55:24]: 0x87244413  |  . $ D .
```

STM32F103C8T6 #5

```
Die xy coords: 107872073
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00565467  |  V T g
Lot_num ascii encoded [55:24]: 0x87014725  |  . . G %
```

STM32F103C8T6 #6

```
Die xy coords: 108003159
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00565666  |  V V f
Lot_num ascii encoded [55:24]: 0x87014633  |  . . F 3
```

STM32F103C8T6 #7

```
Die xy coords: 107872086
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00575566 | W U f
Lot_num ascii encoded [55:24]: 0x87092524 | . . % $
```

STM32F103C8T6 #8

```
Die xy coords: 107741015
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00545171 | T Q q
Lot_num ascii encoded [55:24]: 0x87231927 | . # . '
```

STM32F103C8T6 #9

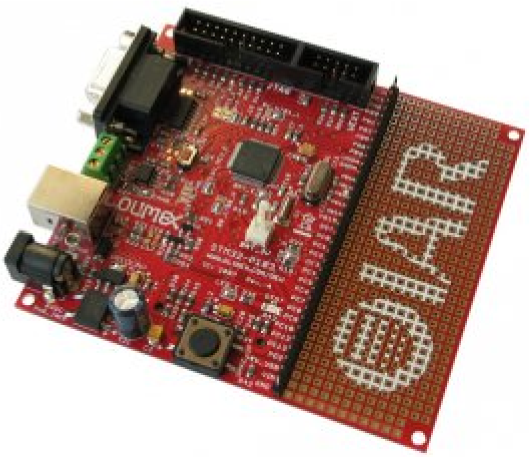
```
Die xy coords: 108396369
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00574867 | W H g
Lot_num ascii encoded [55:24]: 0x87142756 | . . ' V
```

STM32F103C8T6 #10

```
Die xy coords: 108003154
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00545267 | T R g
Lot_num ascii encoded [55:24]: 0x87195634 | . . V 4
```

STm32F103RBT6

Olimex P103 #1



```
Die xy coords: 108068919
Wafer Number: 48
Lot_num ascii encoded  [23:0]: 0x00353650  |  5 6 P
Lot_num ascii encoded [55:24]: 0x43105156  |  C . Q V

FLASH.. TOTAL REPORTED: 131072 USED: 101376 FREE: 29696
```

Olimex P103 #2

```
Die xy coords: 107741012
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00555285  |  U R .
Lot_num ascii encoded [55:24]: 0x87237267  |  . # r g

FLASH.. TOTAL REPORTED: 131072 USED: 103628 FREE: 27444
```

STM32F103ZET

144pinner with 512kB flash and 64kB ram

```
stm32id
Die xy coords: 97779505
Wafer Number: 78
Lot_num ascii encoded  [23:0]: 0x00383341  |  8 3 A
Lot_num ascii encoded [55:24]: 0x51137004  |  Q . p .

free
FLASH.. TOTAL REPORTED: 524288 USED: 51200 FREE: 473088
```

BPill N1

(it has got 128kB).

```
stm32id
Die xy coords: 107806536
Wafer Number: 113
Lot_num ascii encoded  [23:0]: 0x00564989  |  V I .
Lot_num ascii encoded [55:24]: 0x67255444  |  g % T D

free
FLASH.. TOTAL REPORTED: 65536 USED: 51200 FREE: 14336
```

BPill N2

(it has got 128kB).

```
stm32id
Die xy coords: 107741009
Wafer Number: 117
Lot_num ascii encoded  [23:0]: 0x00535783  |  S W .
Lot_num ascii encoded [55:24]: 0x81174113  |  . . A .

free
FLASH.. TOTAL REPORTED: 65536 USED: 51200 FREE: 14336
```

Bluepill

marked as STM32F103C8T6

```
Die xy coords: 108068690
Wafer Number: 114
Lot_num ascii encoded  [23:0]: 0x00525366  |  R S f
Lot_num ascii encoded [55:24]: 0x67145016  |  g . P .

FLASH.. TOTAL REPORTED: 65536 USED: 51200 FREE: 14336
```

GD32F103C8T6

In private board, not a Blue Pill. (

Device descriptor request failed

```
GD32F103-supplied-data
Flash Bytes: 65536
[31:00] $3439CB2E | 4 9 . . | <--Hex char detected
[63:32] $0F383438 | . 8 4 8 | <--Hex char detected
[95:64] $38393733 | 8 9 7 3 |

GD32F103 Data
$1FFFF7E0  = $00140040
$1FFFF7E8  = $3439CB2E
$1FFFF7EC  = $0F383438
$1FFFF7F0  = $38393733
```

Good Bluepill #1

Same marking as #2 below “991KA 93 MYS 903” but 1/2 the flash reported.

```
Die xy coords: 107806545
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00525680  | R V .
Lot_num ascii encoded [55:24]: 0x87192455  | . . $ U

FLASH.. TOTAL REPORTED: 65536 USED: 65536 FREE: 0
```

Good bluepill #2

“991KA 93 MYS 903”

```
Die xy coords: 108068680
Wafer Number: 113
Lot_num ascii encoded  [23:0]: 0x00565366  | V S f
Lot_num ascii encoded [55:24]: 0x67133045  | g . 0 E
```

FLASH.. TOTAL REPORTED: 131072 USED: 51200 FREE: 79872

POS Bluepill

Requires reconfiguring GDB, does not work with Windows STM debug at all.

```
Die xy coords: 74414100
Wafer Number: 47
Lot_num ascii encoded  [23:0]: 0x000E0014  |  . . .
Lot_num ascii encoded [55:24]: 0x170707E2  |  . . . .

FLASH FREE: 0
```

Note

the total lack of Ascii data is telling and the FLASH FREE = 0 is calculated so something is seriously different.

CKS32F103C8T6

Chinese clone

```
Die xy coords: 16985099
Wafer Number: 42
Lot_num ascii encoded  [23:0]: 0x004D3512  |  M 5 .
Lot_num ascii encoded [55:24]: 0x004E4B31  |  . N K 1

Free
FLASH.. TOTAL REPORTED: 131072 USED: 51200 FREE: 79872
```

Note

The Chinese CKS32F103C8T6 isn’t using ASCII for the lot numbers, but hexadecimal and they’re identical to 276745 below.

CKS32F103C8T6

Chinese clone

```
Die xy coords: 276745
Wafer Number: 42
Lot_num ascii encoded  [23:0]: 0x004D3512  |  M 5 .
Lot_num ascii encoded [55:24]: 0x004E4B31  |  . N K 1
```

```

FLASH.. TOTAL REPORTED: 131072 USED: 51200 FREE: 79872
```

STM32F103C8T6

```
Die xy coords: 107741012
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00565283  | V R .
Lot_num ascii encoded [55:24]: 0x87242040  | . $   @

FLASH.. TOTAL REPORTED: 65536 USED: 51200 FREE: 14336
```

Bluepill

```
Die xy coords: 107741008
Wafer Number: 73
Lot_num ascii encoded  [23:0]: 0x00485572  | H U r
Lot_num ascii encoded [55:24]: 0x87243926  | . $ 9 &
```

Bluepill

```
Die xy coords: 108265297
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00515782  | Q W .
Lot_num ascii encoded [55:24]: 0x87061344  | . . . D
```

Bluepill

```
Die xy coords: 108068681
Wafer Number: 73
Lot_num ascii encoded  [23:0]: 0x00545682  | T V .
Lot_num ascii encoded [55:24]: 0x87055015  | . . P .
```

Bluepill

This one didn’t want to program with st-flash as in the supplied script. Programs fine with openocd

```
Die xy coords: 108134227
Wafer Number: 113

```

```
Lot_num ascii encoded  [23:0]: 0x00544872  |  T H r
Lot_num ascii encoded [55:24]: 0x67041446  |  g . . F
```

Bluepill

```
Die xy coords: 108003154
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00565388  |  V S .
Lot_num ascii encoded [55:24]: 0x87093153
```

STM32F103CBT6

```
Die xy coords: 108199730
Wafer Number: 52
Lot_num ascii encoded  [23:0]: 0x00353254  |  5 2 T
Lot_num ascii encoded [55:24]: 0x57012556  |  W . % V
```

GD32VF103 RISC-V

Manufactured by Gigadevice with 128kB Flash and 32kB Ram at 108MHz, this RISC-V core uses most of the STM32F10x peripherals

See also

STM32F103C8 V/S GD32VF103Cx

Features

Core

- **RISC-V** Open Design Core, **NOT ARM**.

SVD Summary

- 46 peripherals
- 1074 registers
- 7713 bitfields

SVD Named Peripherals

```
ADC0       RCU
ADC1       RTC
AFIO       SPI0
BKP        SPI1
CAN0       SPI2
CAN1       TIMER0
CRC        TIMER1
DAC        TIMER2
DBG        TIMER3
DMA0       TIMER4
```

```
DMA1       TIMER5
EXMC       TIMER6
EXTI       USART0
FMC        USART1
FWDGT      USART2
GPIOA      UART3
GPIOB      UART4
GPIOC      USBFS_GLOBAL
GPIOD      USBFS_HOST
GPIOE      USBFS_DEVICE
I2C0       USBFS_PWRCLK
I2C1       WWDGT
ECLIC      PMU
```

Interesting Features

- Instruction profiling for performance measurement using RISC-V “instructions execute” and “instruction clock” registers
- Totally different interrupt peripheral to STM32F10x
- Two ADC’s
- One DAC
- The RISC-V processors have a 12 bit (plus-or-minus 2048) byte conditional branch range. The usual assembly language trick for getting around this limitation is to invert the sense of the test and conditionally branch over an unconditional branch with more offset bits.

Mecrisp-Quintus

Matthias Koch has released his *experimental* port of Mecrisp-Quintus Forth for the GD32VF103 RISC-V MCU

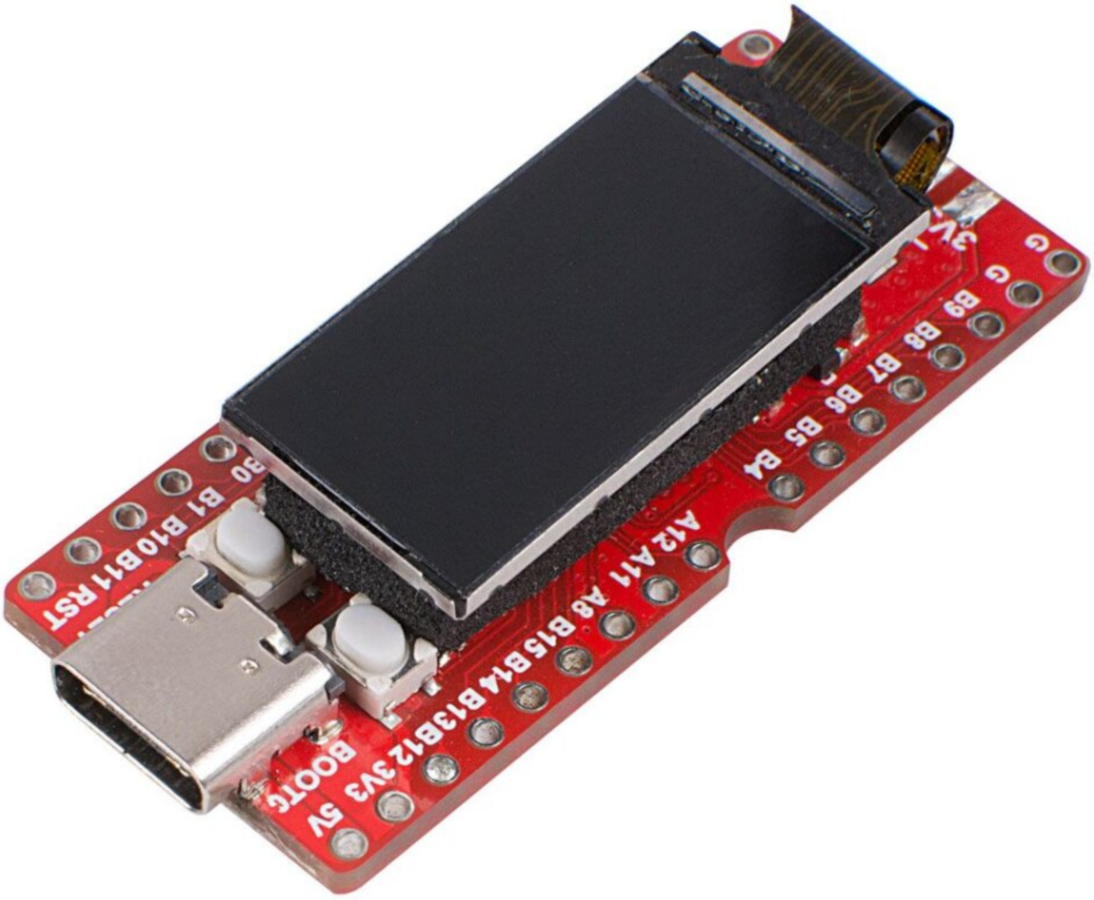
Download it here:
https://sourceforge.net/projects/mecrisp/files/mecrisp-quintus-0.26-experimental.tar.gz/download

Premade Boards

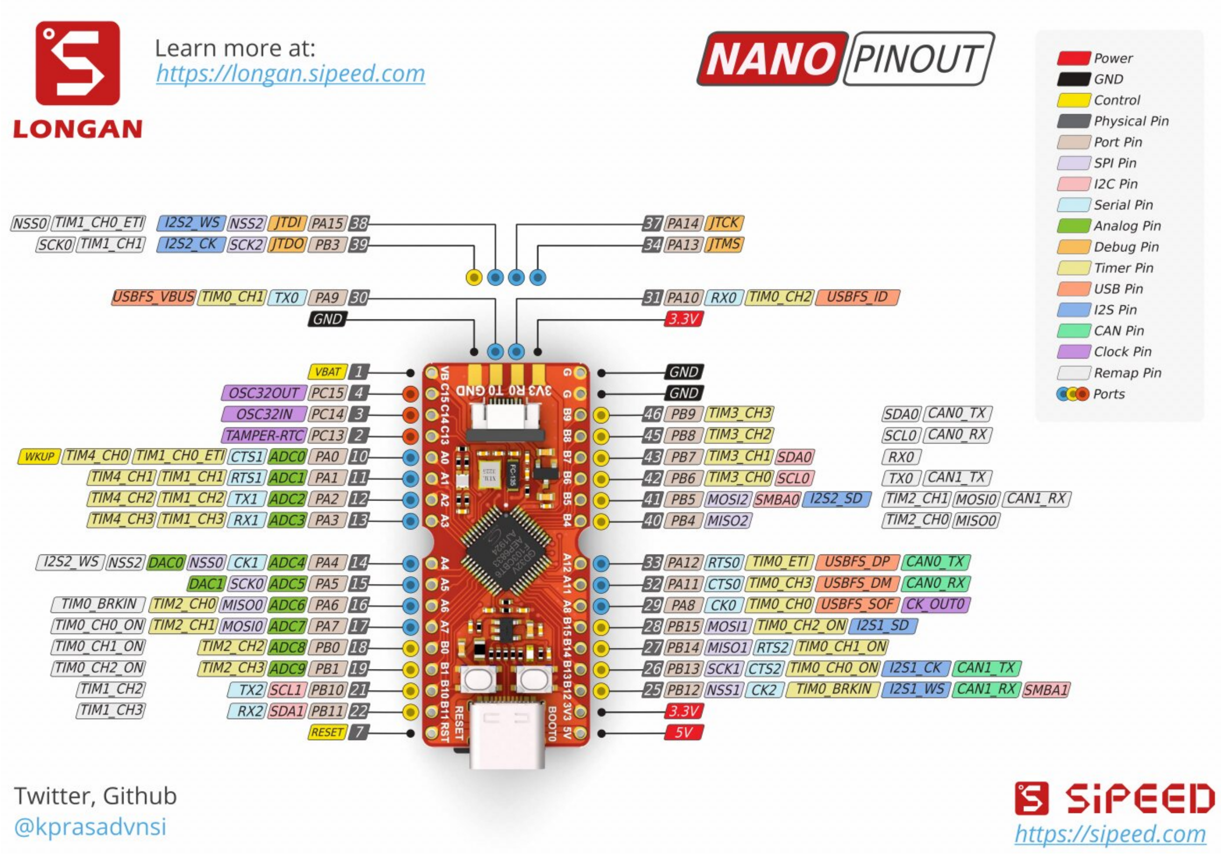
The Longan-Nano

Longan Nano is a $5 development board based on GD32VF103CBT6 MCU with RISC-V 32-bit core of GigaDevice.

- Sold by Seeed comes with a 0.96inch 160x80 IPS RGB LCD and an acrylic transparent case.
- Chip: GD32VF103, 128kB Flash, 32kB Ram at 108MHz.
- Board, 160x80 IPS RGB LCD, USB “C” socket, micro SD slot, Boot-0 and Reset button etc.


RST
B11
B10
B1
5V
3V3
BOOT0
B12
B13
B14
B15
A8
A11
A12
B4
B5
B6
B7
B8
B9
G
G

Pinouts


LONGAN
Learn more at:
https://longan.sipeed.com
NANO PINOUT
Power
GND
Control
Physical Pin
Port Pin
SPI Pin
I2C Pin
Serial Pin
Analog Pin
Debug Pin
Timer Pin
USB Pin
I2S Pin
CAN Pin
Clock Pin
Remap Pin
Ports
NSS0 TIM1_CH0_ETI I2S2_WS NSS2 JTDI PA15 38
SCK0 TIM1_CH1 I2S2_CK SCK2 JTDO PB3 39
37 PA14 JTCK
34 PA13 JTMS
USBFS_VBUS TIM0_CH1 TX0 PA9 30
GND
31 PA10 RX0 TIM0_CH2 USBFS_ID
3.3V
VBAT 1
OSC32OUT PC15 4
OSC32IN PC14 3
TAMPER-RTC PC13 2
WKUP TIM4_CH0 TIM1_CH0_ETI CTS1 ADC0 PA0 10
TIM4_CH1 TIM1_CH1 RTS1 ADC1 PA1 11
TIM4_CH2 TIM1_CH2 TX1 ADC2 PA2 12
TIM4_CH3 TIM1_CH3 RX1 ADC3 PA3 13
I2S2_WS NSS2 DAC0 NSS0 CK1 ADC4 PA4 14
DAC1 SCK0 ADC5 PA5 15
TIM0_BRKIN TIM2_CH0 MISO0 ADC6 PA6 16
TIM0_CH0_ON TIM2_CH1 MOSI0 ADC7 PA7 17
TIM0_CH1_ON TIM2_CH2 ADC8 PB0 18
TIM0_CH2_ON TIM2_CH3 ADC9 PB1 19
TIM1_CH2 TX2 SCL1 PB10 21
TIM1_CH3 RX2 SDA1 PB11 22
RESET 7
GND
GND
46 PB9 TIM3_CH3 SDA0 CAN0_TX
45 PB8 TIM3_CH2 SCL0 CAN0_RX
43 PB7 TIM3_CH1 SDA0 RX0
42 PB6 TIM3_CH0 SCL0 TX0 CAN1_TX
41 PB5 MOSI2 SMBA0 I2S2_SD TIM2_CH1 MOSI0 CAN1_RX
40 PB4 MISO2 TIM2_CH0 MISO0
33 PA12 RTS0 TIM0_ETI USBFS_DP CAN0_TX
32 PA11 CTS0 TIM0_CH3 USBFS_DM CAN0_RX
29 PA8 CK0 TIM0_CH0 USBFS_SOF CK_OUT0
28 PB15 MOSI1 TIM0_CH2_ON I2S1_SD
27 PB14 MISO1 RTS2 TIM0_CH1_ON
26 PB13 SCK1 CTS2 TIM0_CH0_ON I2S1_CK CAN1_TX
25 PB12 NSS1 CK2 TIM0_BRKIN I2S1_WS CAN1_RX SMBA1
3.3V
5V
3V3 R0 T0 GND
VB C15 C14 C13 A0 A1 A2 A3 A4 A5 A6 A7 B0 B1 B10 B11 RST
G G B9 B8 B7 B6 B5 B4 A12 A11 A8 B15 B14 B13 B12 3V3 5V
RESET
BOOT0
Twitter, Github
@kprasadvnsi
SiPEED
https://sipeed.com

Notes:

Manually taken from the `schematic`, so double check in case of errors.

LCD (P1)

160x80 IPS RGB LCD, doesn’t seem have any inbuilt fonts.

GPIO	Signal
Hardwired	LedA (backlight 2V8)

GPIO	Signal
PB1	Reset
PB10	RS
PA7	SDA
PA5	SCL
PB2	CS

LEDS

Colour	GPIO
Red	GPIOC_13
Green	GPIOA_1
Blue	GPIOA_2

Terminal

USART 0 BASE $40013800

Connect a 3.3V terminal bridge to T0 and R0 of the board:

Board	**USB/serial dongle**
3V3	3V3
R0	TXD
T0	RXD
GND	GND

SD Card (J2)

Signal	**PIN**	**Connects to**	**GPIO**
DAT2	1	N/C	
DAT3/CS	2	SPI_CS_TF	PB12
CMD/MOSI	3	SPI1_MOSI	PB15 plus 10K pullup to 3V3
CLK	5	SPI1_SCLK	PB13

Signal	PIN	Connects to	GPIO
DAT0/MISO	7	SPI1_MISO	PB14 plus 10K pullup to 3V3
DAT1	8	N/C	
PAD4	9	GND	
PAD3	12	GND	
PAD2	11	GND	
PAD1	10	GND	

Header (P2)

Pin	GPIO	Description
1		GND
2		GD_UART0_TX
3		GD_UART0_RX

Pin	GPIO	Description
4		3V3
5	PA13	JTMS
6	PA14	JTCK
7	PA15	JTDI
8	PB3	JTDO

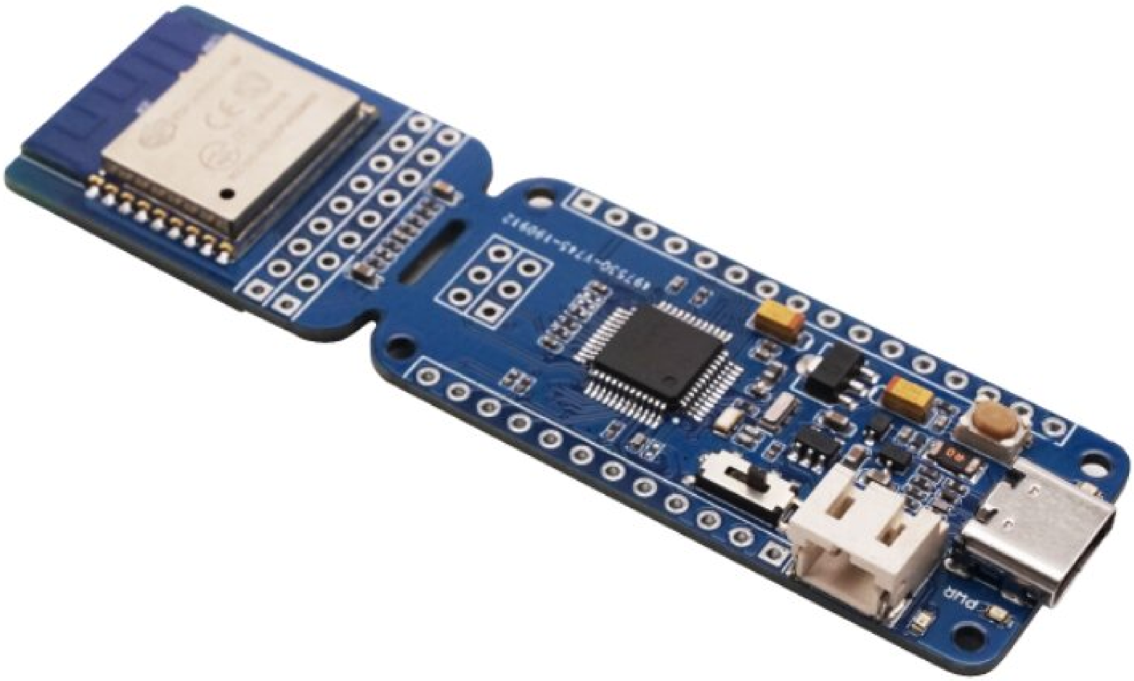
Unused GPIO’s

```
PA3   PB4
PA4   PB5
PA6   PB6
PA8   PB7
PA9   PB8
PA9   PB9
PA10  PB10
PA11  PB11
PA12
```

Wio Lite RISC-V (GD32VF103) - With ESP8266

Wio Lite RISC-V is a feather form factor RISC-V development board Based on GD32VF103, with the onboard ESP8266 Wio Core, it also features WiFi function.

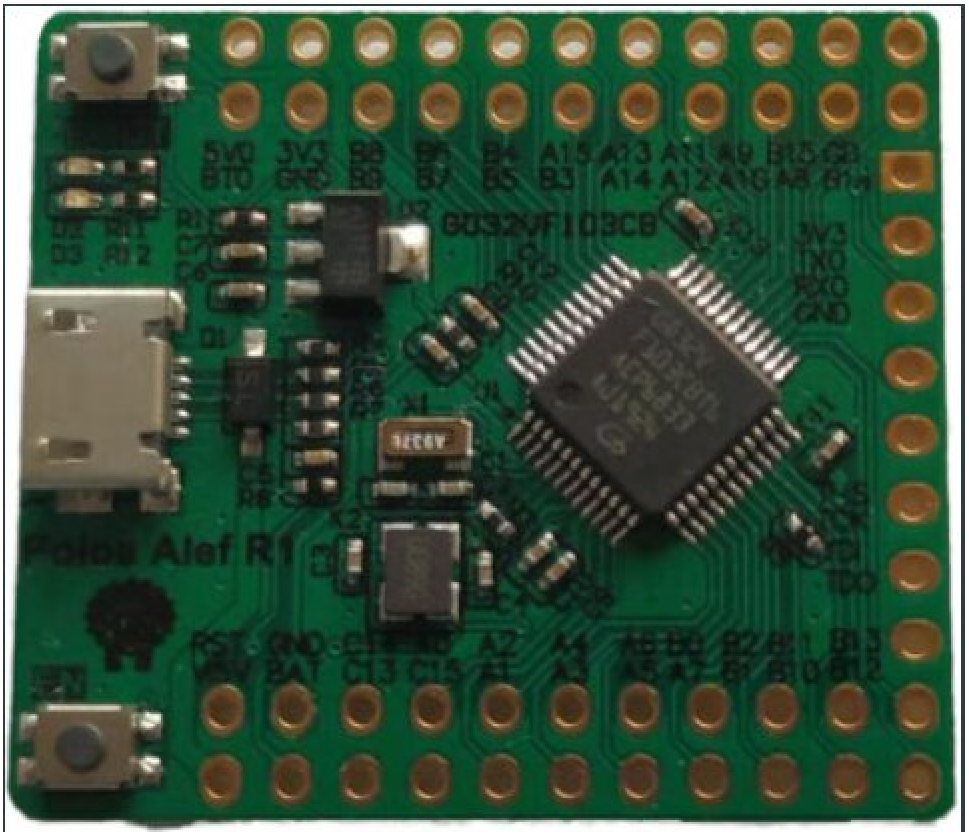
https://www.seeedstudio.com/Wio-Lite-RISC-V-GD32VF103-p-4293.html $6.90 USD



Polos GD32V Alef

Polos GD32VF103 Alef Board – RISC-V MCU Board with 128KB Flash 32KB RAM

https://www.analoglamb.com/product/polos-gd32v-alef-board-risc-v-mcu-board/ $2.99 USD



Flashing the Mecrisp-Quintus binary

The GD32VF103 has a bootloader bug that makes it return the wrong page size and page count for its internal flash. While this is merely an annoyance for uploads (we get extra data), it breaks downloads completely since only half the pages are erased and written. (Every other page is left untouched when we use the reported page size of 2048.)

Nucleisys (the core designer) has a fork of dfu-utils with a fix for this bug [1]. and this patch is a cleaned-up version of that. I’ve added a new quirk that allows custom fixup code to run after the memory layout is read but before it’s used, so much of this code is applicable to other devices that may have similar bugs.

https://github.com/riscv-mcu/gd32-dfu-utils/commit/c2c8215061b08146e6a4c8c22bc57fae95c656

GD Tools for Linux are required to flash!

https://github.com/riscv-mcu/gd32-dfu-utils

```
"description": "GD32V FLASH TOOLS",
"name": "tool-gd32vflash",
"system": "linux_x86_64",
"url": "https://github.com/riscv-mcu/gd32-dfu-utils",
"version": "0.1.0"
```

Flashing

Erasure is required first or a Mecrisp-Quintus update probably won’t run or may have odd behaviour. Erasing is not documented in the dfu-util documentation, for fear of people bricking their (generic) devices but I doubt if bricking is possible in this case.

```
dfu-util -a 0 -d 28e9:0189 -s 0x8000000:mass-erase:force -D
mecrisp-quintus-gd32vf103cb.bin

dfu-util: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future
dfu-util release!!!
Opening DFU capable USB device...
ID 28e9:0189
Run-time device DFU version 011a
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuERROR, status = 10
dfuERROR, clearing status
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 011a
Device returned transfer size 2048
GD32 flash memory access detected
Device model: GD32VF103CB
Memory segment (0x08000000 - 0801ffff)(rew)
```

```
Erase size 1024, page count 128
Performing mass erase, this can take a moment
Setting timeout to 35 seconds

Downloading to address = 0x08000000, size = 18816
Download        [=========================] 100%        18816
bytes
Download done.
File downloaded successfully
```

Now press the reboot button

```
Mecrisp-Quintus 0.26 for RISC-V 32 IMC on GD32VF103CB by
Matthias Koch
```

Connect a Terminal

Use a [USB/3.3v Dongle like this`](#) like one of these.

- power is supplied via the USB connector. Board needs a power/reset after flashing.
- serial comms is connected as below at 115200 baud :

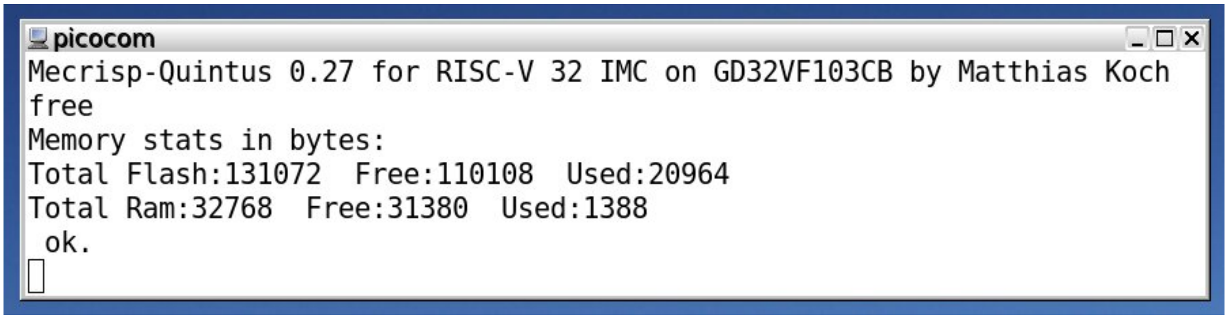
Longan-nano	usb/3.3v serial dongle
To	RX
Ro	TX
0v	0v

Mecrisp-Quintus Greeting Message

```
Mecrisp-Quintus 0.25 for RISC-V 32 IM on GD32VF103CB by
Matthias Koch
```

Utility Programs Ported So Far

Memstats



```
Mecrisp-Quintus 0.27 for RISC-V 32 IMC on GD32VF103CB by Matthias Koch
free
Memory stats in bytes:
Total Flash:131072  Free:110108  Used:20964
Total Ram:32768  Free:31380  Used:1388
 ok.
```

Electronic Device ID

```
id
UNIQUE_ID[31:0]  0x3641294D |  6 A ) M
UNIQUE_ID[63:32] 0x00123736 |  . . 7 6
UNIQUE_ID[95:64] 0xFFFFFFFF |  . . . .
ok.
```

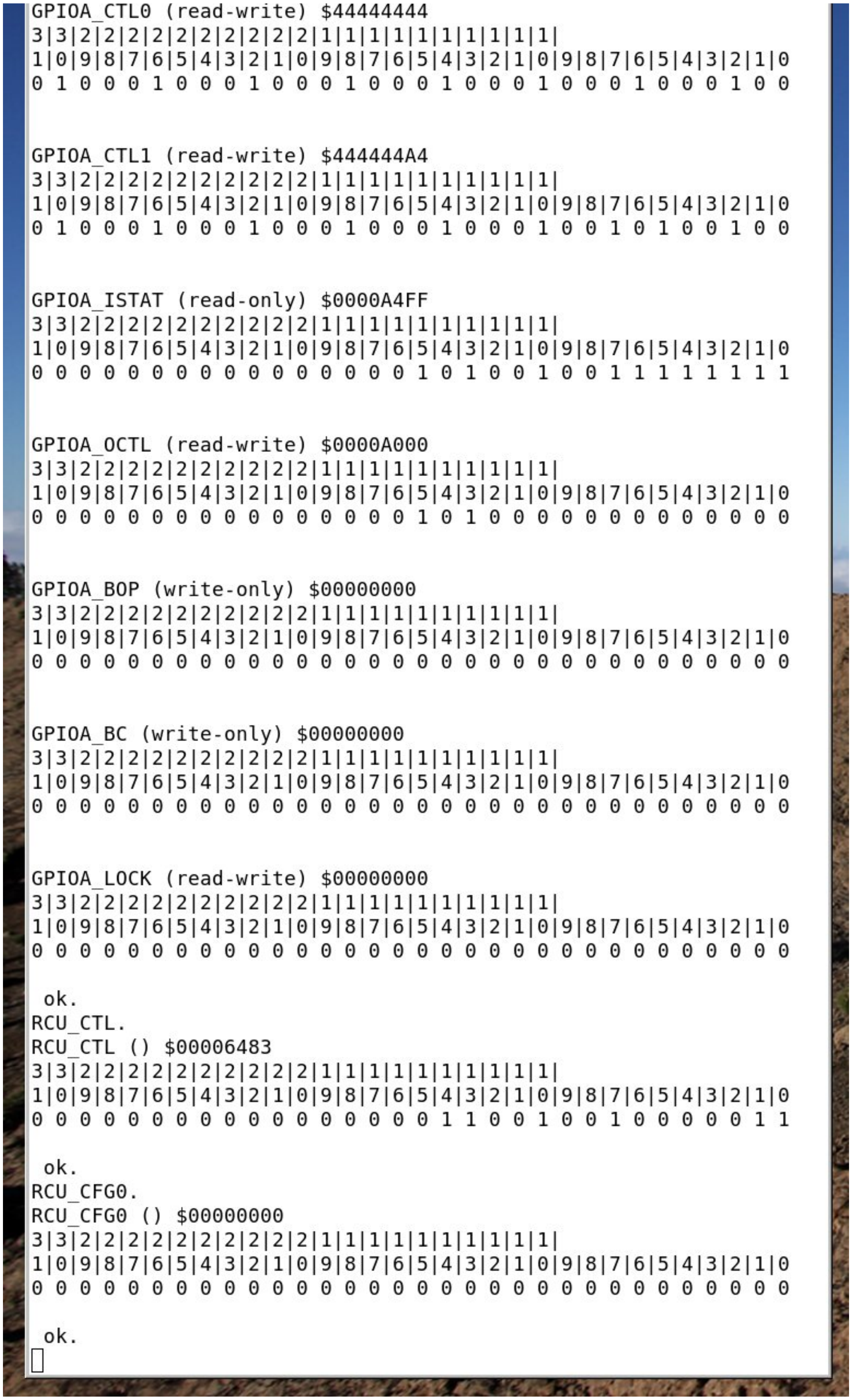
Register Views

Now this device uses the old familiar STM32F103 peripherals, if you know that chip, you’ll recognize the bitfields … BUT *all* the bitfield names have been changed!

There are so many registers that I can’t fit them all in a “development version” of this chip like I can in a 128kB STM32F103.

```
46 peripherals
1074 registers
7713 bitfields
```

```
gpioa.
```



```
GPIOA_CTL0 (read-write) $44444444
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0


GPIOA_CTL1 (read-write) $444444A4
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 1 0 0


GPIOA_ISTAT (read-only) $0000A4FF
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1


GPIOA_OCTL (read-write) $0000A000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0


GPIOA_BOP (write-only) $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


GPIOA_BC (write-only) $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


GPIOA_LOCK (read-write) $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
RCU_CTL.
RCU_CTL () $00006483
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1

 ok.
RCU_CFG0.
RCU_CFG0 () $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
```

Compiling

V-0.26

- A very deep change: Added support for compressed instruction set RV32IMC
- For smaller opcodes, register map was changed in all source files on all targets as compressed instructions are available mostly for x8 to x15:
  - x3 –> x8 TOS
  - x4 –> x9 Data stack pointer
  - x5 –> x14 Unsaved temporary 1
  - x6 –> x15 Unsaved temporary 2
  - x8 –> x3 Loop index
  - x9 –> x4 Loop limit
- Many changes and improvements everywhere, which surely introduced new bugs: If something crashes on 0.26 and works fine on 0.25, please report.
  - Added linux-rv32imc target
  - Switched HX8K and GD32VF103 to IMC Forth core
  - Added FIFO to the receive side of the Icebreaker UART
  - HX8K target is now very similiar to Icebreaker
  - Fixed a few elements left on the stack in fixpt-math-lib-tests.fs
  - Updated toolchain calls to use riscv64-linux-gnu-* available as package “binutils-riscv64-linux-gnu” in Debian 10
- Uses a different toolset to V0.25.

I Installed riscv64-binutils-2.33.1,1 for FreeBSD 12.1 and just had to edit the mecrisp-quintus-0.26-experimental/mecrisp-quintus-source/gd32vf103cb/Makefile like so

```
# ARMGNU?=riscv64-linux-gnu
ARMGNU?=/usr/local/bin/riscv64-unknown-freebsd12.0
```

V-0.25

If you want to cross compile Matthias GD32VF103CB release, grab the toolchain here, it works on both FreeBSD and Linux being statically linked.

```
https://sourceforge.net/projects/mecrisp/files/mecrisp-quintus-toolchain.tar.gz/download
```

Downloads

`72 MHz clock at 115200 baud`

`104 MHz clock at 460800 baud`

`Memstat`

`Words4`

`Device electronic signature`

Fixed Point (s31.32)

Fixed Point (s31.32) allows Mecrisp-Stellaris to compute results including decimal places using microprocessors lacking a “Floating Point Unit”.

Fixpoint numbers are stored (n-comma n-whole) and can be handled like signed double numbers			**Examples**
d-	(df1 df2 - - df3)	Addition of two fixpoint numbers	9,12 7,13 d+ f. 16,24999999976716935634613037109375 ok.
d+	(df1 df2 - - df3)	Subtraction of two fixpoint numbers	9,12 7,13 d- f. 1,9899999999068677425384521484375 ok.
f/	(df1 df2 - - df3)	Division of two fixpoint numbers	9,12 7,13 f/ f. 1,2791023841127753257751464843750 ok.
f*	(df1 df2 - - df3)	Multiplication	9,12 7,13 f* f. 65,02559999795630574226379394531250 ok.
hold<	(char - -)	Adds character to pictured number output buffer from behind	

Fixpoint numbers are stored (n-comma n-whole) and can be handled like signed double numbers			**Examples**
f#S	(n-comma1 - - n-comma2)	Adds 32 comma-digits to number output	
f#	(n-comma1 - - n-comma2)	Adds one comma-digit to number output	9,12 f# f. 90,11999999987892806529998779296875 ok.
f .	(df - -)	Prints a fixpoint number with 32 fractional digits	9,12 7,13 f/ f. 1,27910238411277532577514648437500 ok.
f.n	(df n - -)	Prints a fixpoint number with n fractional digits	9,12 7,13 f/ 3 f.n 1,279 ok.
number	(Counted-String-Address - - 0)	Tries to convert a string to a number.	
	(cstr-addr - - n 1)		

Fixpoint numbers are stored (n-comma n-whole) and can be handled like signed double numbers	**Examples**
(cstr-addr - - n-low n-high 2)	

Fixpoint numbers are stored (n-comma n-whole) and can be handled like signed double numbers.

For fixpoint-numbers you can use d+ and d- as usual, and do division and multiplication with f/ and f*.

Examples

Note

Results starting with * are using Unix bc (arbitrary-precision arithmetic language and calculator) on a 64 bit FreeBSD system.

Display a Number

```
9,12 f. 9,11999999987892806529998779296875  ok.
```

Adds One Comma-digit To Number Output

```
9,12 f# f.
  90,11999999987892806529998779296875  ok.
```

Removes Anything Before Comma

```
1111,1234 f#S f. 0,12339999992400407791137695312500  ok.
```

Addition

```
9,12 7,13 d+ f.
  16,24999999976716935634613037109375  ok.
  * 16.25
```

Subtraction

```
9,12 7,13 d- f.
  1,98999999999068677425384521484375  ok.
  * 1.99
```

Division

```
9,12 7,13 f/ f.
  1,27910238411277532577514648437500  ok.
  * 1.27910238429172510518
```

Multiplication

```
9,12 7,13 f* f.
  65,02559999795630574226379394531250  ok.
  * 65.0256
```

Display Only Three Decimal Places

```
9,12 7,13 f/ 3 f.n
  1,279  ok.
```

s31.32 Insights

Andrew is a Mathematician and was very kind to email me some insight into s31.32 which I have reproduced here. Thanks Andrew!

```
from:  Andrew Palm
date:  Fri, May 4, 2018
subject: Re: s31.32
```

Essentially, a s31.32 number is just a 64-bit signed integer (twos-complement), which is a double in M-S Forth. If the stack has two singles (s1 s2) that represent a double, the s1 single is interpreted as the lower 32 bits, s2 the higher 32 bits. If this double is interpreted as an s31.32, then s1 is the fractional part and s2 is the whole number part. Note that Matthias calls the fractional part the “comma part” because those crazy Europeans use a comma instead of a decimal point. ;-> In other words, we imagine a decimal point (or a comma) between the fractional and whole parts.

Now converting from a single to a double or an s31.32 and back is confusing. If you want to convert an ordinary single s to an ordinary double, you do this: If s is 0 or positive, put a 0 on top of it on the stack. If s is negative, put a -1 on top of it on the stack (because of twos-complement). This is because we put s in the lower 32 bits. On the other hand, if you want to convert a single to an s31.32, you do a 0 swap, putting s on top of a zero on the stack, no matter what the sign of s is. This is because in this case you are putting s in the high 32 bits as the whole part and there is a zero fractional part. It took me a while to sort this out!

To add or subtract s32.32 values you just treat them as doubles and use d+ or d-. To multiiply or divide them you do not use double multiplication or division. Instead you use f* and f/. In the case of multiplication, what goes on behind the scenes is as follows: If you want to multiply the two s31.32 values x and y, first multiply them as doubles to get a 128 bit result x*y, then discard the bottom 32 bits (by right shifting 32 times, of course) and finally keep the bottom 64 bits from that. Here is why: An s31.32 value x is represented by the double value a where x = a/(2^32). If we multiply x by another s31.32 value y represented by the double b, so y = b/(2^32), then x*y = [a*b/(2^32)]/(2^32). That is, a*b/(2^32) is the double value representing x*y. If you do division, we get x/y = [(a/b)*(2^32)]/(2^32) so we take the double division result a/b and multiply it by 2^32 (left shift 32 times) to get the s31.32 representation of x/y.

If you search for “fixed point math” or “fixed point muliplication” or something similar you will get many hits. Including “s31.32” results in almost no hits because it is a special case of fixed point. When I did some DSP on a dsPIC33F I used s0.15, usually called Q15 or Q0.15. This is 16 bit signed integers with the lowest 15 bits all fractional, and no whole part. It is nice for signal work, as it represents numbers between -1 (represented by -32768) and 1 - (2^16) (represented by 32767). If the values get too small or too large, you divide them or multiply them by a gain value to avoid overflow, etc. I think I read somewhere that in the early days of cell phones, they used Q15 for the DSP math. There are often processor instructions to facilitate doing fixed point.

As an example, see: https://sestevenson.wordpress.com/fixed-point-multiplication/

See also: http://www.digitalsignallabs.com/downloads/fp.pdf It is kind of “mathy” in its tone, but seems otherwise OK. The s31.32 representation is called A(31, 32) in the document. The (fairly) standard notation is Q31.32.

Well, I hope this helps a bit.

Cheers, Andy

See also

A sunrise and sunset calculator

The First Rule of Forth

Like the film “Fight Club”, The first rule of Forth, is to not talk about Forth!

I read this interesting article recently which does echo my own experience since starting Forth in 2014.

The Polarizing Effect

Source: https://github.com/narke/colorForth/raw/master/cf2019_colorForth.pdf Page:13 and reproduced in part below.

IMO Forth is a polarizing language because it can dramatically increase productivity in software development. This has several effects :

1. Non-Forth programmers feel uncomfortable when a Forth programmer produces a program in a half or a quarter of the time that they would take. Nobody likes to be made to feel like an idiot.
2. Forth programmers have a significant advantage, so they naturally love “…one of the best-kept secrets in the computing world.”
3. Programming in Forth is fun it is creative rather than mechanical work
4. Forth is different and can have a steep un-learning curve

Warning

People either love Forth or hate Forth there is wide distance between the two extremes

An anecdote

In the late 1990’s I was working on a three month contract, programming an LCD drive in C.

There were four special constraints :

1. The interface was I2C and the power consumption was critical, so only the minimum information must be sent, that is required to update the LCD
2. There were two LCD driver chips, one for the left half of the display, the other for the right
3. The whole display is mounted upside down the graphics mode x,y = (0,0) coordinate is in the bottom right hand corner.
4. The 8x8 character glyphs have to be rotated by 180 degrees.

The interface is similar to ANS Forth’s TYPE and AT-XY, but in C.

To solve this problem I first added a simple talker program, written in C that connected the devices serial port to its I2C bus and memory something like PEEK and POKE, I2Cread and I2Cwrite.

Then I used SwiftForth to create the other side of the talker interface and words to display characters and the memory map in the device that was used to store the current LCD data. I also a word to rotate the font data and save it as a C file.

Having developed the program in Forth on a PC, testing all the while on the actual ARM target device, Itranslated the Forth program to C. Since I knew that I would be doing this I did not use any fancy Forth features such as CREATE …. DOES> or EVALUATE , and I kept the Forth word names compatible with C no “% & * + - . > < ” etc. characters.

The timescale for this was roughly three weeks of fun Forth development, about a week to convert to C. Then I cleaned up the code a bit, improved the documentation and so on, and after about 6 weeks I presented the finished code to my colleague an experienced and competent C programmer. He was surprised that I had completed the work so quickly, but was embarrassed that I had shown that his original time estimate was so wrong. No one likes to feel that they are inferior to anybody else, or that their career based on C is maybe not the most productive. My contract was not extended.

> What’s New ? <

Flashing Mecrisp-Stellaris onto a STM32: Methods

There are a number of methods to upload a binary image of Mecrisp-Stellaris to a STM32 MCU. These include SWD, JTAG and Serial Bootloader.

If this is your first time, and youre in a *hurry*, checkout the Quickstart page ?

Flashing Methods

Difficulty Score, 1 is easiest.

Difficulty	Flashing Method	Software	Required Hardware	URL
1	Serial Bootloader	stm32loader.py	3.3v USB Serial Dongle	`stm32loader.py`
1	St-flash (FreeBSD)	st-flash (FreeBSD)	SWD	Freebsd, OpenBSD use the port or package ” devel/stlink “
1	df-util	df-util	USB	http://dfu-util.sourceforge.net/
2	SWD	OpenOCD	JTAG or SWD	https://sourceforge.net/projects/openoc
3	GDB	GDB over OpenOCD	JTAG or SWD	https://www.gnu.org/s/gdb/

Serial Bootloader

Note

See www.st.com/resource/en/application_note/cd00167594.pdf for which serial ports support bootloading on your STM32F MCU.

Your board will need one or two Boot Option Jumpers. On some Discovery boards only BOOT0 may be present as BOOT1 is internally tied to logic high inside the MCU and not available externally.

Howto flash Mecrisp-Stellaris using the Bootloader

1. Download the latest Mecrisp-Stellaris tarball: http://mecrisp.sourceforge.net/
2. Extract the binary for your MCU, in this case it’s mecrisp-stellaris-stm32fxxx.bin
3. Hook up your 3.3v USB Serial Dongle to the USART 1 (only USART-1 supports the serial bootloader on a STM32F103xx (PA9-TX, PA10-RX) pins on your cheap Chinese target board . You can usually power the board from the dongle.
4. Set the boot pin jumpers. Boot 0 to HIGH or 1. BOOT 1 will stay LOW or 0.
5. Run something like this command ” python ./stm32loader.py -p /dev/cuaU0 -evw ./mecrisp-stellaris-stm32fxxx.bin “
6. Set both boot pins to LOW, reset the power to the target
7. Fire up your serial terminal and talk Forth to your MCU/Board.

Special note to use the Bootloader on a Discovery Board

Fast and simple, requires the use of Boot Option Jumpers, stm32loader.py and a serial 3.3v/USB dongle. Note the STM32F0 Discovery Board has a BOOT0 pin next to a VDD pin. A - E below describe how to use the inbuilt STM32F0 Serial Bootloader on a STM32F0 Discovery board, but many other Discovery boards have the same Bootloader Jumper Methods.

A. Have a serial connection to PA9/10 (This is the default Mecrisp-Forth serial connection)
B. Fit 0.1” pitch Option Jumper on the STM32F0 Discovery Board between ‘BOOT0’ and ‘VDD’. Borrow one from the nearby ST-LINK jumper pins.
C. Press the STM32F0 Discovery Board RESET button
D. Run the following CLI command “python ./stm32loader.py -p /dev/cuaU0 -evw ./mecrisp-stellaris-stm32f051.bin”
E. Remove the Option Jumper from ‘BOOT0’ and ‘VDD’ and replace it on the ST-LINK header. DONE! Faster than SWD and easy, no need to find a usb cable etc.

A Unix friendly Bootloader: stm32loader.py

- Author: Ivan A-R <ivan@tuxotronic.org>
- With hacky error recovery by Gordon Williams <gw@pur3.co.uk>
- Project page: http://tuxotronic.org/wiki/projects/stm32loader
- The version here `stm32loader.py` was kindly cleaned up by McKinnon Ryall vatrat.dev@gmail.com.

```
# python ./stm32loader.py -h
Usage: ./stm32loader.py [-hqVewvrX] [-l length] [-p port] [-b baud] [-a addr] [file.bin]
   -h          This help
   -q          Quiet
   -V          Verbose
   -e          Erase
   -w          Write
   -v          Verify
   -X          Reset after
   -r          Read
   -l length   Length of read
   -p port     Serial port (default: first USB-like port in /dev)
   -b baud     Baud speed (default: 115200)
   -a addr     Target address
   -s n        Skip writing N bytes from beginning of the binary (does not affect start
address)
   -k          Change PCLK frequency to make USB stable on Espruino 1v43 bootloaders

   ./stm32loader.py -e -w -v example/main.bin
```

```
stm32loader.py -p /dev/cuaU16 -evw mecrisp-stellaris-stm32f103.bin
```

```
Reading data from mecrisp-stellaris-stm32f103.bin
Bootloader version 0x22
Chip id 0x410, STM32F1, performance, medium-density
Writing 20352 bytes to start address 0x8000000
Write 256 bytes at 0x8000000
Write 256 bytes at 0x8000100
Write 256 bytes at 0x8000200
...
Read 256 bytes at 0x8004F00
Verification OK
```

```
Mecrisp-Stellaris RA 2.3.9 for STM32F103 by Matthias Koch
 ok.
 ok.
 ok.
```

Details:

- The serial bootloader is stored in the internal boot ROM memory (system memory) of STM32 devices. It is programmed by ST during production and cant be wiped or overwritten. Its main task is to download the application program to the internal Flash memory through one of the available serial peripherals (USART, CAN, USB, I2C, SPI, etc.). A communication protocol is defined for each serial interface, with a compatible command set and sequences.
- Selection is using the two “boot” jumpers on the MCU board. Set Boot 1 = LOW (0) and Boot 2 = HIGH (1) before using the bootloader. After bootloading set them both to LOW (0) to run the binary you just uploaded and power cycle the board.
- A ‘bootloader’ program needs to be run on the PC, I’ve used stm32loader.py without any problems on a Shenzhen LC mini STM32FC8T6 board from China.
- Which USART do I use on a chip/board with multiple USARTS ? See STM Application note cd00167594 for your MCU. Mecrisp-Stellaris usually uses USART-1
- See this table for serial port pins and connections to find your most likely Bootloader port connections.
- After Mecrisp-Stellaris has uploaded, set the two boot jumpers to LOW and reset the power. You should then be able to use a serial terminal to talk Forth to the board using exactly the same serial port on the target board.
- Mecrisp-Stellaris takes 6 seconds to upload its 23KB image with stm32loader.py and seems to work everytime on a cheap $9 Shenzhen LC mini STM32FC8T6 board from China.

Warning

Requires a USB/3.3v cable or dongle. You can’t use RS232, if you do you may blow up your STM32 chip.

Boot Pin Functions

Pins BOOT0 and BOOT1 select how the STM32 starts

BOOT 1	BOOT 0	Boot mode

Pins BOOT0 and BOOT1 select how the STM32 starts

X	0	User Flash memory (application)
0	1	System memory (bootloader)
1	1	Embedded SRAM

Saving a on chip binary

Saving 645kB of flash to “saved-binary.bin”

```
./stm32loader.py -r -a  0x08000000 -l 0x10000 -p /dev/ttyUSB0 saved-binary.bin
```

Stlink

Howto flash Mecrisp-Stellaris using Stlink

1. Download the latest Mecrisp-Stellaris tarball: http://mecrisp.sourceforge.net/
2. Extract the binary for your MCU, in my case it’s: mecrisp-stellaris-stm32f051.bin
3. Install Stlink
    - FreeBsd and OpenBsd: devel/stlink
    - Linux: You may need to compile it with Linux Debian as they don’t seem to have a stlink.deb. See https://github.com/texane/stlink

4. Connect your Discovery board to your PC using a usb cable, a led for the board power should light.

Note: Root or sudo will be required for the erase and write steps below

5. Erase the flash on the board by running

```
st-flash erase
```

It’s advised to press the reset button on the board before the next step.

6. Now flash the board with the Mecrisp-Stellaris binary, remember to use the binary for *your* MCU

```
st-flash write mecrisp-stellaris-stm32f051.bin 0x08000000
```

7. Connect a 3.3v-USB dongle to your PC and make sure it works in a serial terminal emulator such as “Picocom”, “Minicom”, “Cutecom” etc by connecting the 3.3v TX and RX wires and observing that keys are echoed to the screen when hit. See this table for which pins to use for your board/chip. Connection config is: 115200 baud, 1 stop bit, no parity bit, and no handshaking, no xon/off. Below is a picocom example but your /dev/cuaU0 device will probably be different to mine.

```
picocom -b 115200 /dev/cuaU0 –imap lfcrlf,crcrlf –omap delbs,crlf
```

8. Connect the TX, RX and GND wires from the 3.3v-USB dongle to your Discovery board, which pins are used depends on the board/MCU so check the pin numbers in the “Connecting a Serial Terminal” Section in this documentation.
9. When all is working, pressing the reset button on your Discovery Board will result in a similar message being sent to your screen.

```
Mecrisp-Stellaris 2.3.6 with M0 core for STM32F051 by Matthias Koch
```

FINISHED! at this point Mecrisp-Stellaris Forth is running on your Microcomputer.

While Picocom is great for a quick test, I recommend that you use E4thcom as it is specifically made for programming and uploading files to a Forth Micocomputer.

DF-UTIL

Df-util is a host side implementation of the DFU 1.0 and DFU 1.1 specifications of the USB forum. DFU is intended to download and upload firmware to/from devices connected over USB. It ranges from small devices like micro-controller boards to mobile phones. Using dfu-util you can download firmware to your DFU-enabled device or upload firmware from it.

Example usage

Flashing a .dfu (special DfuSe format) file to the device

```
$ dfu-util -a 0 -D /path/to/dfuse-image.dfu
```

Reading out 1 KB of flash starting at address 0x8000000

```
$ dfu-util -a 0 -s 0x08000000:1024 -U newfile.bin
```

Flashing a binary file to address 0x8004000 of device memory and ask the device to leave DFU mode

```
$ dfu-util -a 0 -s 0x08004000:leave -D /path/to/image.bin
```

Unhandled Interrupt 00000003

Warning

I have observed that failure to fully erase the Flash memory prior to flashing Mecrisp-Stellaris often results in endless “Unhandled Interrupt 00000003” messages, this can happen with df-util because the erase-flash command is undocumented for fear of people bricking their devices.

How To Erase Using Df-util

https://forum.mystorm.uk/t/how-to-clear-all-flash/373 (erikolson)

Here’s how I flash with erase:

```
dfu-util -a 0 -d 0483:df11 -s 0x8000000:mass-erase:force -D mecrisp-stellaris-stm32l433.bin
-t 1024
```

You can “upload” the flash to your desktop to see if you do have unwanted flash content:

```
#!/bin/bash
# verify upload
rm current.bin
dfu-util -a 0 -d 0483:df11 -s 0x8000000 -U current.bin
od -x mecrisp-stellaris-stm32l433.bin > in.txt
od -x current.bin > out.txt
sdiff -w 216 -s in.txt out.txt | more
```

This is also useful if you suspect corruption which happens if the upload is not a 4-byte multiple.

SWD

- Uses the SWD pins on the MCU and STLINKv2 firmware (factory installed on STM32 “Discovery Boards”)
- Requires a USB/SWD programmer (Every STM32 “Discovery Board” is a USB to SWD programmer and comes with “STLINK”)
- Requires a USB port on the PC
- Use STLINKv2 only, avoid all old STLINKv1 SWD boards like the plague
- Can be used for On Chip Debugging with GDB

Howto flash Mecrisp-Stellaris using SWD

To be written ..

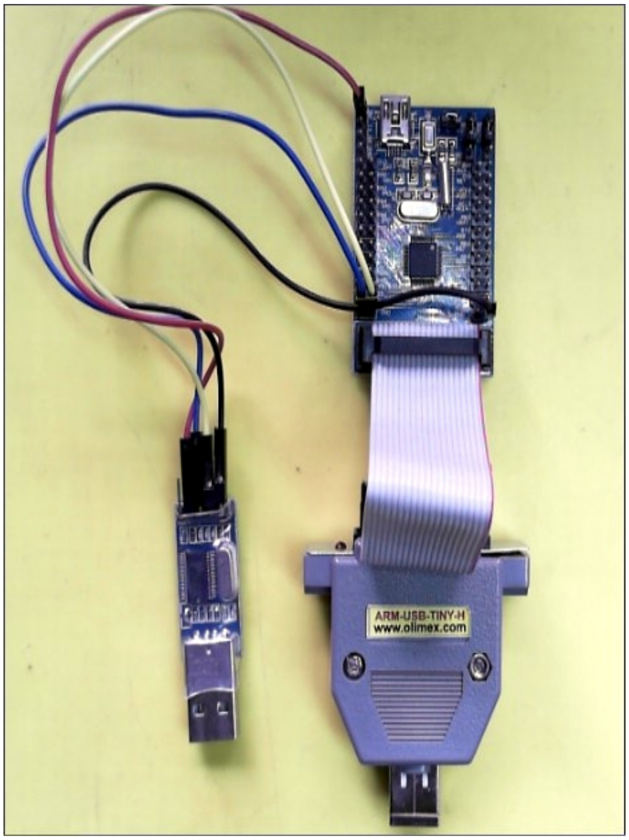
JTAG

- Older less used method
- Uses the JTAG pins on the MCU
- Requires a USB/JTAG Dongle and a JTAG connector on the MCU pcb.
- Requires a USB port on the PC
- The author uses a ‘Olimex ARM-USB-TINY-H” USB to 20 pin IDC JTAG connector on both Linux and FreeBSD
- Can be used for On Chip Debugging with GDB

Howto flash Mecrisp-Stellaris using JTAG

- Board: STM (Shenzen Mini-Arm board) STM32F103C8T6
- FreeBSD OS is used in this example, so your file locations, names and versions may differ to mine if you use Linux etc.
- Olimex ARM-USB-TINY-H JTAG dongle
- OpenOCD is used to Flash the STM32F103C8T6
- Usb to 3.3v dongle for talking serial to Forth on the mini-stm32f103 board via a serial terminal. PL2303 chipsets are recommended being Unix friendly.
- Picocom is used in this example as the serial terminal

The Hardware used



Open a Root Xterm to run OpenOCD. Leave it running until MCU is flashed:

```
root@Gronk:/home/tp # openocd -f /usr/local/share/openocd/scripts/interface/ftdi/olimex-arm-usb-tiny-h.cfg -f /usr/local/share/openocd/scripts/target/stm32f1x.cfg
Open On-Chip Debugger 0.9.0 (2016-02-03-08:48)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "jtag". To override use 'transport select <transport>'.
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
jtag_ntrst_delay: 100
none separate
cortex_m reset_config sysresetreq
Info : clock speed 1000 kHz
Info : JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32f1x.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver: 0x1)
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Open a Telnet session to localhost 4444, which is how the user talks to OpenOCD

```
tp@Gronk:~/projects/programming-languages/forth/mecrisp-stellaris/mecrisp-stellaris-2.2.4/stm32f103-ra# telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger

> poll
background polling: on
```

```
TAP: stm32f1x.cpu (enabled)
target state: halted
target halted due to undefined, current mode: Thread
xPSR: 00000000 pc: 00000000 msp: 00000000

> reset halt
JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
JTAG tap: stm32f1x.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver: 0x1)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00003b6a msp: 0x20000330
```

Check how much Flash you have

```
>  flash probe 0
device id = 0x20036410
flash size = 64kbytes
flash 'stm32f1x' found at 0x08000000

>  flash probe 1
flash bank 1 does not exist
flash bank '#1' is out of bounds

> flash probe 2
flash bank 2 does not exist
flash bank '#2' is out of bounds

>
```

Check if the Flash is erased

```
> flash erase_check 0
device id = 0x20036410
flash size = 64kbytes
successfully checked erase state
        #  0: 0x00000000 (0x400 1kB) not erased
        #  1: 0x00000400 (0x400 1kB) not erased
        #  2: 0x00000800 (0x400 1kB) erased
        #  3: 0x00000c00 (0x400 1kB) erased
        #  4: 0x00001000 (0x400 1kB) not erased
        ... all not erased
        # 56: 0x0000e000 (0x400 1kB) not erased
        # 57: 0x0000e400 (0x400 1kB) erased
        # 58: 0x0000e800 (0x400 1kB) erased
        # 59: 0x0000ec00 (0x400 1kB) erased
        # 60: 0x0000f000 (0x400 1kB) erased
        # 61: 0x0000f400 (0x400 1kB) erased
        # 62: 0x0000f800 (0x400 1kB) erased
        # 63: 0x0000fc00 (0x400 1kB) erased
>
```

Mass erase all the Flash

```
> stm32f1x mass_erase 0
stm32x mass erase complete
```

Check the Flash is really erased

```
> flash erase_check 0
successfully checked erase state
        #  0: 0x00000000 (0x400 1kB) erased
        ... all erased
        # 63: 0x0000fc00 (0x400 1kB) erased
>
```

Flash Mecrisp-Stellaris onto the STM32F103C8T6

```
> flash write_image /home/tp/projects/programming-languages/forth/mecrisp-stellaris/mecrisp-stellaris-2.2.4/stm32f103-ra/mecrisp-stellaris-stm32f103.bin 0x08000000
wrote 20336 bytes from file /home/tp/projects/programming-languages/forth/mecrisp-stellaris/mecrisp-stellaris-2.2.4/stm32f103-ra/mecrisp-stellaris-stm32f103.bin in 0.652892s (30.418 KiB/s)
>
```

Check the Flash, how much space did Mecrisp-Stellaris use up ?

```
flash erase_check 0
successfully checked erase state
        #  0: 0x00000000 (0x400 1kB) not erased
        #  1: 0x00000400 (0x400 1kB) not erased
        #  2: 0x00000800 (0x400 1kB) not erased
        #  3: 0x00000c00 (0x400 1kB) not erased
        #  4: 0x00001000 (0x400 1kB) not erased
        #  5: 0x00001400 (0x400 1kB) not erased
        #  6: 0x00001800 (0x400 1kB) not erased
        #  7: 0x00001c00 (0x400 1kB) not erased
        #  8: 0x00002000 (0x400 1kB) not erased
        #  9: 0x00002400 (0x400 1kB) not erased
        # 10: 0x00002800 (0x400 1kB) not erased
        # 11: 0x00002c00 (0x400 1kB) not erased
        # 12: 0x00003000 (0x400 1kB) not erased
        # 13: 0x00003400 (0x400 1kB) not erased
        # 14: 0x00003800 (0x400 1kB) not erased
        # 15: 0x00003c00 (0x400 1kB) not erased
        # 16: 0x00004000 (0x400 1kB) not erased
        # 17: 0x00004400 (0x400 1kB) not erased
        # 18: 0x00004800 (0x400 1kB) not erased
        # 19: 0x00004c00 (0x400 1kB) not erased
        # 20: 0x00005000 (0x400 1kB) erased
        ... all erased
        # 63: 0x0000fc00 (0x400 1kB) erased
>
```

If you want to Flash by calling OpenOCD from a script, instead of a open OpenOCD terminal and manual Telnet session, you can use this command instead.

- This command flashes, verifies and exits, but does not erase any pre existing Forth Dictionary.

```
root@Gronk:/home/tp/projects/programming-languages/forth/mecrisp-stellaris/mecrisp-stellaris-2.2.4/stm32f103-ra # openocd -f /usr/local/share/openocd/scripts/interface/ftdi/olimex-arm-usb-tiny-h.cfg -f /usr/local/share/openocd/scriptsget/stm32f1x.cfg -c "program mecrisp-stellaris-stm32f103.bin verify exit 0x08000000"

Open On-Chip Debugger 0.9.0 (2016-02-03-08:48)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "jtag". To override use 'transport select <transport>'.
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
jtag_ntrst_delay: 100
none separate
cortex_m reset_config sysresetreq
Info : clock speed 1000 kHz
Info : JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32f1x.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver: 0x1)
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32f1x.bs tap/device found: 0x16410041 (mfg: 0x020, part: 0x6410, ver:
0x1)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00004d92 msp: 0x2000038c
** Programming Started **
auto erase enabled
Info : device id = 0x20036410
Info : flash size = 64kbytes
wrote 20480 bytes from file mecrisp-stellaris-stm32f103.bin in 1.155165s (17.314 KiB/s)
** Programming Finished **
** Verify Started **
verified 20336 bytes in 0.311900s (63.672 KiB/s)
** Verified OK **
shutdown command invoked
```

Connect a terminal to the board and start talking in Forth.

- The mini-stm32f103 board uses pins PA9 (TX) and PA10 (RX), and these will need to be wired to your usb to 3.3v cable or dongle.
- OpenOCD terminals may need to be exited or the board reset switch pressed

```
tp@Gronk:~/projects/programming-languages/forth/mecrisp-stellaris/mecrisp-stellaris-
2.2.4/stm32f103-ra# picocom -b 115200 /dev/cuaU5 --imap lfcrlf,crcrlf --omap delbs,crlf --send-
cmd "ascii-xfr -s -l100"

picocom v2.1

port is         : /dev/cuaU5
flowcontrol     : none
baudrate is     : 115200
parity is       : none
databits are    : 8
stopbits are    : 1
escape is       : C-a
local echo is   : no
noinit is       : no
noreset is      : no
nolock is       : no
send_cmd is     : ascii-xfr -s -l100
receive_cmd is  : rz -vv -E
imap is         : crcrlf,lfcrlf,
omap is         : crlf,delbs,
emap is         : crcrlf,delbs,

Type [C-a] [C-h] to see available commands

Terminal ready
Mecrisp-Stellaris RA 2.2.4 for STM32F103 by Matthias Koch
  ok.
  ok.
```

See also

PROBLEMS ? See This Page

Forth Development Applications

I often use the following Open Source software during Forth program development.

Asciio

This perl application allows you to draw ASCII diagrams in a modern (but simple) graphical interface.

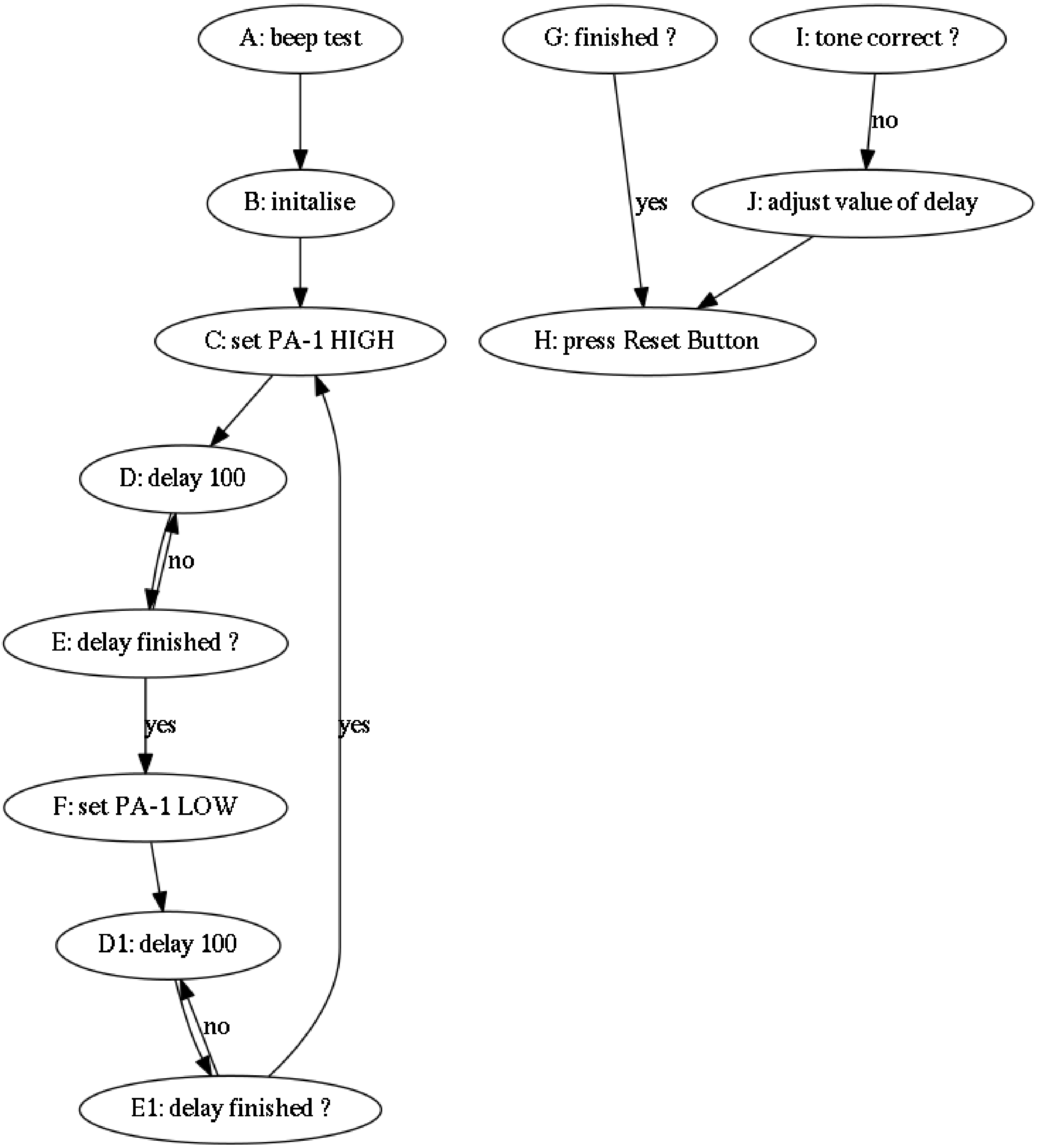
Asciio URL: http://search.cpan.org/dist/App-Asciio/lib/App/Asciio.pm

See my page using Asciio to visualise the Forth Stack

```
.--------------.
|  Data Stack  |
|--------------|
| next element |average @ + average !
| dup          |." data = " .
| dup          |minv @ < if--.
| dup          |              '------> minv ! ." (new minv) "
or DROP after ELSE
| dup          |maxv @ > if---.
| dup          |               '---->  maxv ! ." (new maxv) "
or DROP after ELSE
'--------------'
```

Graphviz

Flowcharting; Directed Graphs


A: beep test
B: initalise
C: set PA-1 HIGH
D: delay 100
no
E: delay finished ?
yes
F: set PA-1 LOW
D1: delay 100
no
E1: delay finished ?
yes
G: finished ?
yes
I: tone correct ?
no
J: adjust value of delay
H: press Reset Button

Why Flowchart ?

When designing new programs I often find it difficult to visualise the complete solution, especially if it is complex. One method that has solved this for me is Graphviz.

Now, as long as I start graphing *something*, (usually the main aims of the program), I find that the smaller parts *always* seem to just fill themselves in and soon I have the overall structure of my initial solution.

Why Graphviz ?

Graphviz http://www.graphviz.org is open source graph visualization software.

Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.

The Graphviz layout programs take descriptions of graphs in a simple text language, and make diagrams in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser. Graphviz has many useful features for concrete diagrams, such as options for colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes.

These are the advantages that are important to me

- Open Source, Graphviz is unlikely to disappear or become unusable
- Text based, so the source is easily versioned in my SCM along with the application source code, keeping them together
- Great documentation
- Beautiful automatic graph layouts
- Very configurable with many features

My method so far when designing a program is to list the main operations and then produce the graph. This is the least complex method I have found which produces the least cluttered graph. I can then play around until I’m sure I have what I want, with Graphviz providing the graph to assist my understanding of what I’m trying to create.

As they say, a picture is worth a thousand words.

Dotme flowchart generator

Because I use so many flowcharts, I have made a flowchart template generator using a shell script called ‘dotme’.

How does it work ?

- download and uncompress dotme.tar.gz somewhere
- configure the three user variable lines in “dotme” to suit your unix system, make it executable and place it in your path
- type ‘dotme’ in any directory you wish to create a flowchart and you will be asked for a flowchart name, enter it. We will call it ‘flowchart’ for this example.
- dotme will then create ‘flowchart.dot’ and a Makefile in the directory so make sure you don’t already have a Makefile in it !
- type ‘make’ and Graphviz will compile the ‘flowchart.dot’ creating a ‘flowchart.png’ which will then be displayed.
- alter the code in ‘flowchart.dot’ to suit your program, run make to display it … repeat until you’re finished.

Requirements:

- Graphviz
- sh
- make
- feh, xli, qiv or any picture viewer

Download The Dotme Tarball

`dotme.tar.gz`

Folding

Foldability is perhaps the most puzzling feature of Mecrisp-Stellaris Forth, hopefully this article helps to clarify it.

x-foldable

Are flags which tells the compiler how many constant inputs are necessary for a definition to produce deterministic outputs on the stack already at compile time.

Flag	Description
0-foldable:	constants variables [‘] [char]
1-foldable:	?dup drop negate
2-foldable:	+ - * swap nip
3-foldable:	rot
4-foldable:	d+ d- 2swap

Sometimes, for very simple definitions, the RA cores can automatically determine that something is foldable, but for a complicated piece like the sine function, declaring it as 2-foldable will help to remove a lot of cycles when a double meets a sine at compile time.

Foldability is for anything which works on the stack without any side effects, and generates constant outputs when it encounters a specified number of constant inputs. Mostly mathematical functions, bit mask tools, stack jugglers and the like.

Folding Example

```
: sqrt ( u -- u^1/2 )
  [
  $2040 h, \   movs r0, #0x40
  $0600 h, \   lsls r0, #24
  $2100 h, \   movs r1, #0
  $000A h, \ 1:movs r2, r1
  $4302 h, \   orrs r2, r0
  $0849 h, \   lsrs r1, #1
  $4296 h, \   cmp r6, r2
  $D301 h, \   blo 2f
  $1AB6 h, \   subs r6, r2
  $4301 h, \   orrs r1, r0
  $0880 h, \ 2:lsrs r0, #2
  $D1F6 h, \   bne 1b
  $000E h, \   movs r6, r1
  ]
  1-foldable
;
```

Compiled Code With Folding

see sqrt-of-9-1-foldable

```
2000043C: 3F04  subs r7 #4
2000043E: 603E  str r6 [ r7 #0 ]
20000440: 2603  movs r6 #3
20000442: 4770  bx lr
```

Compiled Code Without Folding

see sqrt-of-9

```
2000045C: 3F04  subs r7 #4
2000045E: 603E  str r6 [ r7 #0 ]
20000460: 2609  movs r6 #9
20000462: B500  push { lr }
20000464: F7FF  bl  20000400  --> sqrt
20000466: FFCC
20000468: BD00  pop { pc }
```

> [What’s New ?](#) <

Forth (embedded) Interactive Alternatives for Cortex M on STM32F

A quick look at other Open Source interactive development systems for **Cortex M on STM32F**, are they any good, do they even compete with Forth ?

Distribution	Website	Last Updated?	Min Flash/Ram to run (KB) ?	Run on Cortex M0	binary ima
Micropython	https://micropython.org/	current V1.1.1	256 16	No	stm32f4disc
eLua	http://www.eluaproject.net	current V0.9	260KB	No	elua_lua_stn
Armpit Scheme	http://armpit.sourceforge.net/	July 22, 2018 V070	60KB	No	armpit_060_

Note

whilst not supporting STM32F**, uLisp looks to be a classy project, and definitely worth a visit: http://www.ulisp.com/. Speed tests have shown uLisp to be about 200x slower than Mecrisp-Stellaris

Micropython

MicroPython v1.9 on 2017-05-26; F4DISC with STM32F407

How does Micropython Work?

Micropython compiles the python source into bytecode, then the interpreter evaluates the bytecode. Using the [Decorator](#) [http://docs.micropython.org/en/v1.9.3/pyboard/reference/speed_python.html#the-viper-code-emitter] it compiles typed python into assembly.

You can write functions in Arm assembly with the Decorator
[http://docs.micropython.org/en/v1.9.3/pyboard/pyboard/tutorial/assembler.html]

Testing Micropython

I installed the stm32f4disc-20170526-v1.9.dfu image on a STM32F4 Discovery following the tutorial at https://github.com/micropython/micropython/wiki/Board-STM32F407-Discovery, which went very smoothly and quickly.

The Micropython bulk USB storage device mounted easily with the Micropython files visible and the terminal was fast and bugfree using Picocom on Linux (freeBSD was buggy however).

Micropython Terminal demo

```
>>> help()
Welcome to MicroPython!

For online help please visit http://micropython.org/help/.

Quick overview of commands for the board:
 pyb.info()    -- print some general information
 pyb.delay(n)  -- wait for n milliseconds
 pyb.millis()  -- get number of milliseconds since hard reset
 pyb.Switch()  -- create a switch object
                  Switch methods: (), callback(f)
 pyb.LED(n)    -- create an LED object for LED n (n=1,2,3,4)
                  LED methods: on(), off(), toggle(), intensity(<n>)
 pyb.Pin(pin)  -- get a pin, eg pyb.Pin('X1')
 pyb.Pin(pin, m, [p]) -- get a pin and configure it for IO mode m, pull mode p
                  Pin methods: init(..), value([v]), high(), low()
 pyb.ExtInt(pin, m, p, callback) -- create an external interrupt object
 pyb.ADC(pin)  -- make an analog object from a pin
                  ADC methods: read(), read_timed(buf, freq)
 pyb.DAC(port) -- make a DAC object
                  DAC methods: triangle(freq), write(n), write_timed(buf, freq)
 pyb.RTC()     -- make an RTC object; methods: datetime([val])
 pyb.rng()     -- get a 30-bit hardware random number
 pyb.Servo(n)  -- create Servo object for servo n (n=1,2,3,4)
                  Servo methods: calibration(..), angle([x, [t]]), speed([x, [t]])
 pyb.Accel()   -- create an Accelerometer object
                  Accelerometer methods: x(), y(), z(), tilt(), filtered_xyz()

Pins are numbered X1-X12, X17-X22, Y1-Y12, or by their MCU name
Pin IO modes are: pyb.Pin.IN, pyb.Pin.OUT_PP, pyb.Pin.OUT_OD
Pin pull modes are: pyb.Pin.PULL_NONE, pyb.Pin.PULL_UP, pyb.Pin.PULL_DOWN
Additional serial bus objects: pyb.I2C(n), pyb.SPI(n), pyb.UART(n)

Control commands:
 CTRL-A        -- on a blank line, enter raw REPL mode
 CTRL-B        -- on a blank line, enter normal REPL mode
 CTRL-C        -- interrupt a running program
 CTRL-D        -- on a blank line, do a soft reset of the board
 CTRL-E        -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
>>>
```

STM32F4 Discovery led tests

```
>>> led2 = pyb.LED(2)
>>> led2.on()          <-- the board led 2 turned on
>>> led2.off()          <-- the board led 2 turned off

>>>  led2.toggle()      <-- toggles the led!
```

Conclusion

Note

Micropython needs a Cortex M4 generally.

- Positives
    - Stable, fast, seems well designed
    - Documantation is excellent
    - Lots of libraries
    - The CTRL-E serial terminal “paste mode”, which is just like a rock solid full screen editor is just *awesome*, I want one for Forth too!
- Not so positives
    - The Flash Image I built was 309KB. The website (https://micropython.org/) says “it is compact enough to fit and run within just 256k of code space and 16k of RAM”. By Contrast Mecrisp-Stellaris only requires a Flash image of 20KB, making it perfect for inexpensive MCUS with as little as 32KB Flash.
    - For some reason the Micropython home website (https://github.com/micropython/micropython) does not offer any prebuilt binary images.
    - Garbage collection is required which means that there will be delay while this is done, possibly a few mS, making this non real time.
    - Manual testing is tedious due to the language constraints, I’ve been spoilt by Forth.

My thanks to Per Ljung for updates and information re Micropython, and Vincent Hamp for taking the time to point out a major typo.

eLua

The binary flashed to a STM32F407VG Discovery easily using st-flash as below:- `st-flash write elua_lua_stm32f407vg.bin 0x08000000`

Then a terminal was connected at 115200 Baud, 8,n,1.

```
picocom -b 115200 /dev/cuaU20
```

Terminal	Board
TX	PD-9
RX	PD-8

eLua Terminal demo

```
eLua# ver
eLua version v0.8
For more information visit www.eluaproject.net and wiki.eluaproject.net

eLua# help
Shell commands:
 exit        - exit from this shell
 help        - print this help
 ls or dir   - lists filesystems files and sizes
 cat or type - lists file contents
 lua [args]  - run Lua with the given arguments
 recv [path] - receive a file via XMODEM. If path is given save it there, otherwise run it.
 cp <src> <dst> - copy source file 'src' to 'dst'
 wofmt       - format the internal WOFS
 ed  [args]  - run 'ed' text editor with the given arguments
 ver         - print eLua versio

 eLua# lua /rom/info.lua
 Press CTRL+Z to exit Lua
 I'm running on platform STM32F4
 The CPU is a STM32F407VG
 The board name is STM32F4DSCY

 eLua# lua
 Press CTRL+Z to exit Lua
 Lua 5.1.4  Copyright (C) 1994-2011 Lua.org, PUC-Rio
 > print ( 355 / 113.0)
 3.141592920354

 > print ( math.sin(355/113/2))
 0.99999999999999
```

eLua Blinky

The following program based on a eLua example file was loaded, and the board led blinked nicely.

```
-- eLua blinking led example, the Hello World of embedded :)

local uartid, invert, ledpin = 0, false

-- STM32F4 Discovery
 ledpin = pio.PD_13
 uartid = uart.CDC

function cycle()
 if not invert then
   pio.pin.sethigh( ledpin )
 else
   pio.pin.setlow( ledpin )
 end
 tmr.delay( 0, 500000 )
 if not invert then
   pio.pin.setlow( ledpin )
 else
   pio.pin.sethigh( ledpin )
 end
 tmr.delay( 0, 500000 )
end

pio.pin.setdir( pio.OUTPUT, ledpin )
print( "Hello from eLua on STM32F4 Discovery" )
print "Watch the red led blinking :)"
print "Press the Disco Reset Button to end this demo.\n"

while 1
```

```
do  cycle()
end
```

Conclusion

- Positives
    - Stable, fast, seems well designed
    - Documantation is very good
    - eLua has a online image builder for 21 targets (http://builder.eluaproject.net/web/start.lua?control=builder&act=index free registration required) but sadly it didn’t have a STM32F4 Discovery build option when I tried it, and there is no binary available on the hompage. A binary has bene built and is available from this url: http://stm32f4.web.fc2.com/elua.html
- Not so positives
    - File upload is via Xmodem, which is painful with Minicom, but there are probably better ways.
    - Requires a 260KB Flash image, this makes it unusable on any MCU with less than 512KB Flash. By Contrast Mecrisp-Stellaris only requires a Flash image of 20KB, making it perfect for inexpensive MCUS with as little as 32KB Flash.
    - Garbage collection is required which means that there will be delay while this is done, possibly a few mS, making this non real time.
    - Manual testing is tedious like Micropython due to the language requirements

Armpit Scheme

Armpit Scheme is an interpreter for the Scheme language (lexically-scoped dialect of Lisp) that runs on RISC microcontrollers with ARM core. The website has a very neat and academic look and feel.

Armpit scheme looked promising when installed on a stm32f4 discovery, but only partially ran. It’s also available for a tm4c1294xl which I haven’t tried yet.

Summary

I would have been interested in Micropython if I hadn’t already seen the Forth light a few years ago.

For instance, if I want to turn a led on or off with Forth, I’ll make a ‘lon’ and ‘loff’ Word that I use during manual testing, and change it to something like ‘overspeed-indicator-led-on’ (or off) when it’s debugged and part of a lower level Word where the naming is everything.

With Microptyhon I’d find ‘l.on()’ and ‘l.off()’ too tedious for on the fly manual typing during testing as it involves double the keys and two shifts. I’d be trapped in a non extensible language again where the problem solution still looks like the same language I started with.

I’m a hardware guy, I’ll have the keyboard balanced on the scope, trying to avoid the hot soldering iron and my coffee as I’m developing, so I want the most for the least when I type.

eLua on Cortex M4 is not a bad effort either, but requires similar resources to Micropython.

Armpit-Scheme is still dead and stuck in 2014.

In summary, I can only find two competitors for Forth, and neither of them will even run on any MCU with less than 256K Flash.

Not to mention lack of speed, ram usage, garbage collection issues and so on …

In the small embedded end of the market, this means that Forth really has no interactive competitors, so *nothing has changed* that I can see.

Forth Books

Warning

There is much in these books that will work with Mecrisp-Stellaris and a lot that won’t work because all Forths are different in some way. If something doesn’t work, don’t assume it is you, research the matter further, ask for help on the irc channel or the forum.

Modern Forth Books

Stephen Pelc’s excellent MPE Forth Book

It has some sections unique to MPE Forth, but this is my preferred Forth reference. Linked here with permission from Mr Pelc.
http://www.mpeforth.com/arena/ProgramForth.pdf

And so Forth

Copyright J.L. Bezemer 2001-04-06 released under the GPL.

I think this is a **excellent Forth Primer**, **`download it from this site (pdf-200KB)`**

Or read it online at
https://thebeez.home.xs4all.nl/ForthPrimer/Forth_primer.html

The two must read *classic* books

- Starting Forth, by Leo Brody, this is THE BOOK to learn how to code in Forth. It’s a bit dated now but still excellent reading.

  HTML : https://www.forth.com/starting-forth/

  Book: http://www.amazon.com/Starting-Forth-Prentice-Hall-Software-Series/dp/0138430799

- Thinking Forth, by Leo Brody This is about problem solving and designing the “Forth Way”.

  PDF, HTML, EBOOK: https://sourceforge.net/projects/thinking-forth/

  BOOK: http://www.amazon.com/Thinking-Forth-Leo-Brodie/dp/0976458705

Forth Book Lists

http://www.pcorner.com/list/FORTH/DIARY.ZIP/DIARY.TXT/

Forth Links

Mecrisp-stellaris

https://sourceforge.net/projects/mecrisp/

Why I Use Forth by Bernd Paysan

https://bernd-paysan.de/why-forth.html

Forth Still Suits Embedded Applications (Tom Napier | Nov 21, 1999)

http://www.electronicdesign.com/boards/forth-still-suits-embedded-applications

The Forth Programming Language - Why YOU should learn it

http://www.hcsw.org/reading/forth.txt

Forth - the LEGO of Programming Languages 1995

http://www.ultratechnology.com/4thlego.htm

Why Forth ?

http://www.complang.tuwien.ac.at/forth/faq/why-forth

Old Tutorial List

http://www.forth.org/tutorials.html

Forths at Github

https://github.com/ForthHub

Space-Related Applications of Forth

http://web.archive.org/web/20101024223709/http://forth.gsfc.nasa.gov/

Forth Variants

Direct threaded code.

In this model, the code field contains machine code instead of a pointer to machine code This is somewhat faster, but typically costs extra bytes for some classes of words. It is most prevalent on 32-bit systems.

Subroutine-threaded code

In this model, the compiler places a jump-to-subroutine instruction with the destination address in-line. This technique costs extra bytes for each compiled reference on a 16-bit system. It is often slower than direct-threaded code, but it is an enabling technique to allow the progression to native code generation.

Forth Versus Other Systems

Some examples.

Riyadh, Saudi Arabia Airport Project

The Forth team of 15 programmers delivered a working system in 18 months, replacing a failed effort that 30 programmers using conventional HLLs had failed to deliver after 3 years. It’s the largest, but not unique. Source: [e.d.rather] 20 Oct 2017, Usenet: comp.language.forth

Copyright © ACS

Forth Based Commercial Products

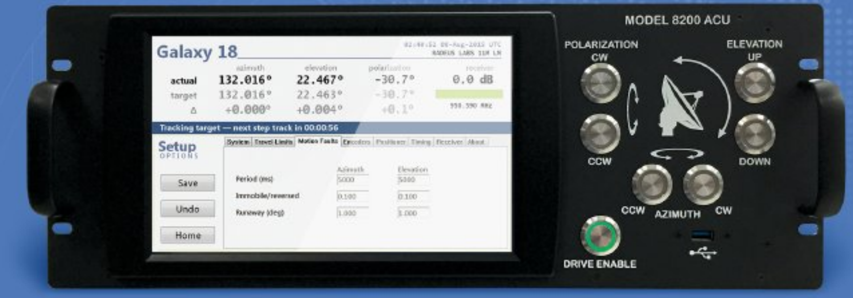
Some examples.

Ground Station Dish Controller

source: [e.d.rather] 21 Oct 2017, Usenet: comp.language.forth

MODEL 8200™

ANTENNA CONTROL SYSTEM


MODEL 8200 ACU
POLARIZATION
CW
CCW
ELEVATION
UP
DOWN
CCW
AZIMUTH
CW
DRIVE ENABLE
Galaxy 18
actual
target
Δ
132.016°
22.467°
-30.7°
0.0 dB
132.016°
22.463°
-30.7°
+0.000°
+0.004°
+0.1°
Tracking target — next step track in 00:00:56
Setup
OPTIONS
Save
Undo
Home
Period (ms)
Immobile/reversed
Runaway (deg)

The Next Generation in Antenna Control

Performance – Flexible tracking modes, intuitive menu layouts and a compact parameter set keep your limited motion antenna applications on point.

Availability – We understand the need for quick delivery. Lean manufacturing methods allow us to ship most systems within 30 days of an accepted order!

World-Class Support – You are never on your own with a Radeus Labs product. The experts at Radeus Labs are standing by if you need help.

RADEUS
LABS, INC.

Mecrisp-Stellaris Videos

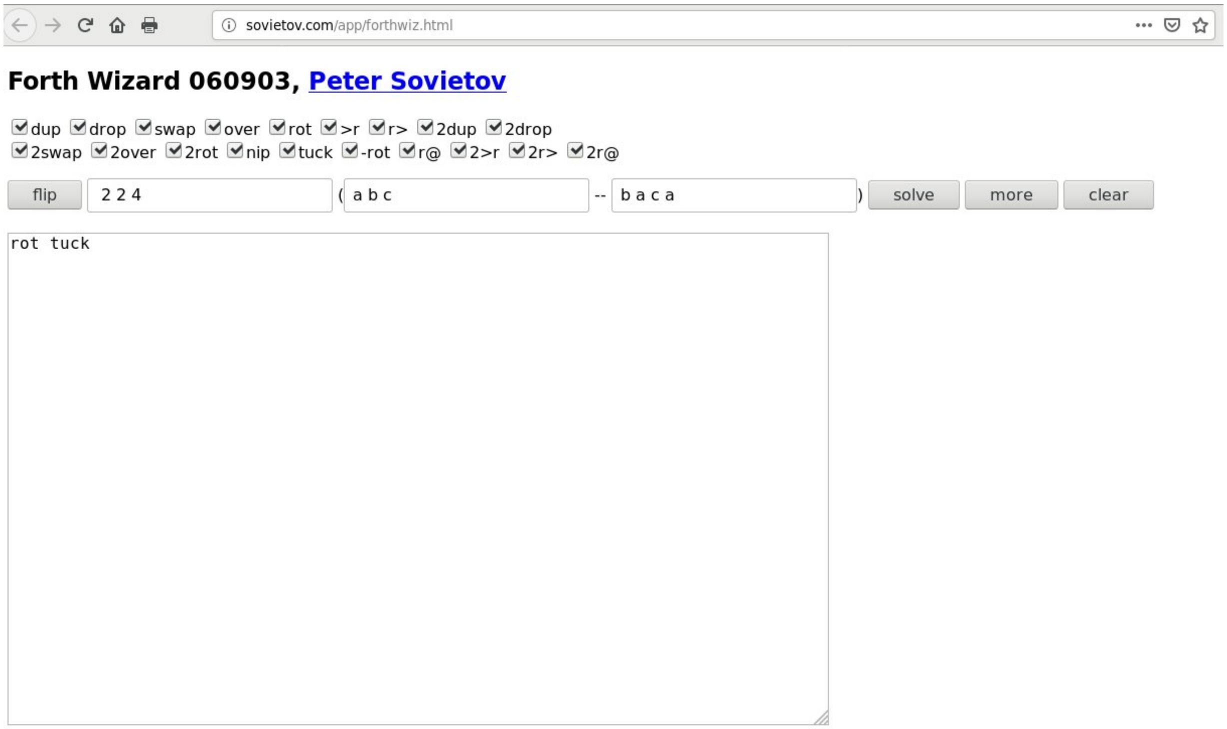
Josh shows some of the fun he has been having with the Mecrisp-Stellaris interactive programming environment

Josh also looks into some of the clever things the Mecrisp-Stellaris compiler does in this clear and fast paced hi-def video. https://www.youtube.com/watch?v=dvTI3KmcZ7I

Forth Wizard

http://sovietov.com/app/forthwiz.html

A Stack Manipulation Tool


sovietov.com/app/forthwiz.html
flip
solve
more
clear

Forth Wizard 060903, Peter Sovietov

☑dup ☑drop ☑swap ☑over ☑rot ☑>r ☑r> ☑2dup ☑2drop
☑2swap ☑2over ☑2rot ☑nip ☑tuck ☑-rot ☑r@ ☑2>r ☑2r> ☑2r@

2 2 4 (a b c -- b a c a)

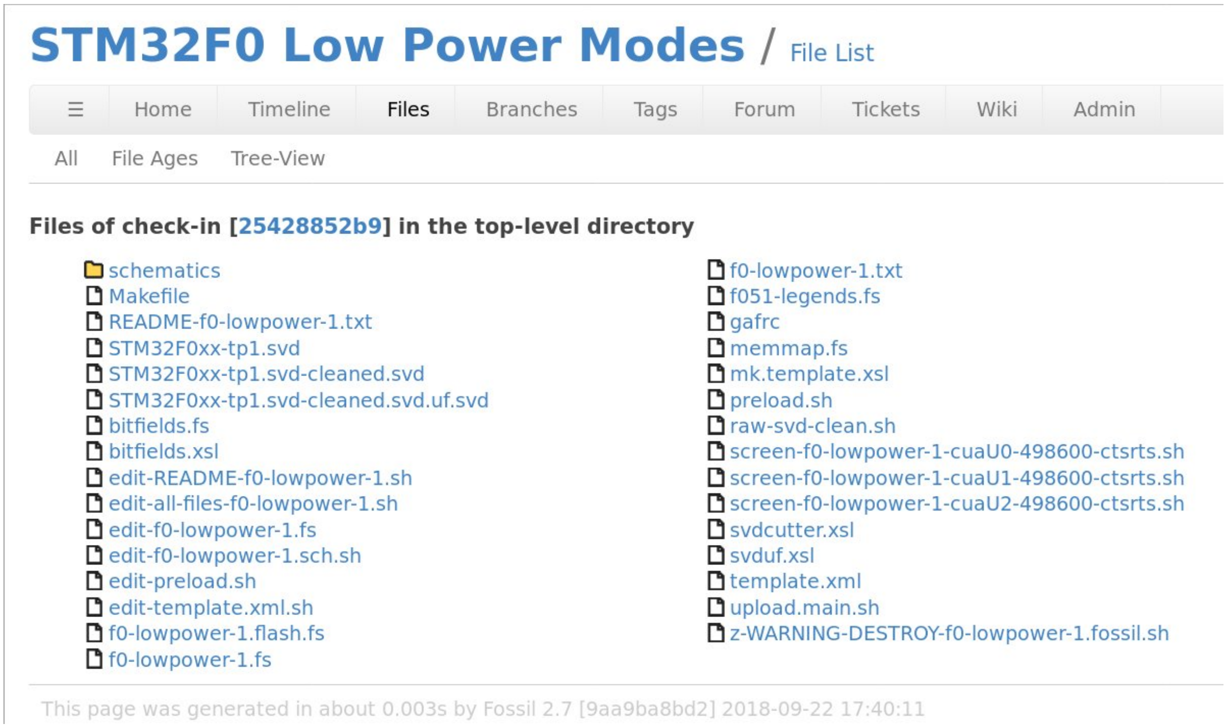
```
rot tuck
```

This handy online tool by Peter Sovietov generates stack operations based on your required outcome. This is the result from the picture above.

```
2 2 4 rot tuck . . . . 2 4 2 2  ok.
```

Fossil Distributed Version Control System

Fossil isn’t a part of Mecrisp-Stellaris but even Forth can take advantage of a decent DVCS.



Files of check-in [25428852b9] in the top-level directory

- schematics
- Makefile
- README-f0-lowpower-1.txt
- STM32F0xx-tp1.svd
- STM32F0xx-tp1.svd-cleaned.svd
- STM32F0xx-tp1.svd-cleaned.svd.uf.svd
- bitfields.fs
- bitfields.xsl
- edit-README-f0-lowpower-1.sh
- edit-all-files-f0-lowpower-1.sh
- edit-f0-lowpower-1.fs
- edit-f0-lowpower-1.sch.sh
- edit-preload.sh
- edit-template.xml.sh
- f0-lowpower-1.flash.fs
- f0-lowpower-1.fs
- f0-lowpower-1.txt
- f051-legends.fs
- gafrc
- memmap.fs
- mk.template.xsl
- preload.sh
- raw-svd-clean.sh
- screen-f0-lowpower-1-cuaU0-498600-ctsrts.sh
- screen-f0-lowpower-1-cuaU1-498600-ctsrts.sh
- screen-f0-lowpower-1-cuaU2-498600-ctsrts.sh
- svdcutter.xsl
- svduf.xsl
- template.xml
- upload.main.sh
- z-WARNING-DESTROY-f0-lowpower-1.fossil.sh

This page was generated in about 0.003s by Fossil 2.7 [9aa9ba8bd2] 2018-09-22 17:40:11

Fossil Quickstart Page:

http://www.fossil-scm.org/index.html/doc/trunk/www/quickstart.wiki

An excellent Fossil Summary page by Brandon Invergo

http://brandon.invergo.net/news/2013-07-11-A-quick-introduction-to-version-control-with-Fossil.html

The Fossil Bbook

http://www.fossil-scm.org/schimpf-book/home

Garbage Collection

C, C++, Objective C, Swift, Forth, Fortran, Basic, Pascal, Rust, Alef, Ada (most implementations) and many other high level programming languages do not have a garbage collector. [andrea.ferro]

Forth Does Not Need Garbage Collection

Forth does not need garbage collection because transient data is pushed on the stack. So by the very nature of the language memory is automatically freed when the word which held the memory exits. You push the desired amount of memory on the stack, then call all of the words which need to access that block of memory. When everything is done it’s freed. It’s just the same as if you malloc(3c) a block on entry to a function and then call free(3c) just before you exit. Forth has the great advantage that you can’t have a memory leak if you honor the principles on which it is founded. [reginald.beardsley]

Glossary

Cornerstone

A Cornerstone is a special Word which can be defined at any point. Calling this word will then erase all definitions added *after* it.

Cornerstone Word for 1 KB Flash pages

```
: cornerstone ( Name ) ( -- )
  <builds begin here $3FF and while 0 h, repeat
  does>   begin dup  $3FF and while 2+   repeat
  eraseflashfrom ;
```

Example

1. Enter the “cornerstone” Word into Flash on your system as above
2. Enter the following, the reply from Mecrisp-Stellaris is indented

```
words
  Address: 00003CE6 Link: 00004000 Flags: 0000FFFF Code:
00003D06 Name: --- Flash Dictionary ---

cornerstone corner-a
: testa 2 2 + . cr ;
words

  Address: 00003CE6 Link: 00004000 Flags: 0000FFFF Code:
00003D06 Name: --- Flash Dictionary ---
  Address: 00004000 Link: 0000405C Flags: 00000000 Code:
00004012 Name: cornerstone
  Address: 0000405C Link: 00004400 Flags: 00000000 Code:
0000406C Name: corner-a
  Address: 00004400 Link: FFFFFFFF Flags: 00000000 Code:
```

```
0000440C Name: testa

corner-a
  Erase block at  00004404  from Flash
  Mecrisp-Stellaris 2.4.5 for TM4C1294 by Matthias Koch

words
  Address: 00003CE6 Link: 00004000 Flags: 0000FFFF Code:
00003D06 Name: --- Flash Dictionary ---
```

Warning

Depending on the Flash implementation of the MCU, results may vary from this example.

Imm

> An immediate value (or simply an immediate or imm) is a piece of data that is stored as part of the instruction itself instead of being in a memory location or a register. Immediate values are typically used in instructions that load a value or performs an arithmetic or a logical operation on a constant.
>
> ARM data processing instructions have 12 bits of space for values in their instruction word. This is arranged as a four-bit rotate value and an eight-bit immediate value.
>
> The 4-bit rotate value stored in bits 11-8 is multiplied by two giving a range of 0-30 in steps of two.

Using this scheme we can express immediate constants such as:

```
0x000000FF
0x00000FF0
0xFF000000
0xF000000F
```

But immediate constants such as:

```
0x000001FE
0xF000F000
0x55550000
```

…are not possible.

An assembler will convert big values to the rotated form. Impossible values will cause an error.

Some assemblers will use other tricks such as using MVN instead of MOV to form the bitwise complement of the required constant. For example the impossible instruction MOV r0,#0xFFFFFFFF could be assembled as MVN r0,#0. [davespace]

<builds does>

Or create does>

Michael Ham (https://galileo.phys.virginia.edu/classes/551.jvn.fall01/primer.htm#create) has called the word pair CREATE…DOES>, the “pearl of Forth”. CREATE is a component of the compiler, whose function is to make a new dictionary entry with a given name (the next name in the input stream) and nothing else. DOES> assigns a specific run-time action to a newly CREATEd word.

Folding

> Constant folding is a well known technique in optimisation. It means that if an operator works on constants the result may be replaced by a constant that is calculated at compile time. In Forth we generalise this

to folding. Folding refers to all words that can be replaced by simpler words in case they receive constant data on the stack. [vanderhorst]

“n-foldable” is mostly used on words that take n arguments as input. When the compiler encounters a n-foldable word, it looks back to see if the word is preceded by n literals (or constants), puts them on the stack, calls the word, and compiles the result as a literal.

Inlining

Inlining means replacing a Forth word with its constituents. This technique is very important in Forth, more so than in other languages, due to the small size of Forth words. Inlining is always a winner in speed, and mostly even also a winner with regard to space.

Even more important is the fact that inlining allows folding to be applied across constituent words. This applies to high level and low level code alike.

Inlining high level code is trivial. A further inlining stage replaces a high level definition that only calls code words, by a code definition which concatenates the code words. [vanderhorst]

Inlining means taking the code of a word, and putting it in place of a call to this word (saving the cost of a call, usually at the cost of memory)

Inlining Example (Classic Kernel, NON -RA)

First the Word “42+ ” is defined and called by “insert-here” which has to branch to “42+ ”. No Inlining is used.

```
: 42+ 42 + ;  ok.

see 42+
20000340: B500  push { lr }
20000342: 362A  adds r6 #2A
20000344: BD00  pop { pc }
 ok

: insert-here 42+ ;  ok.

see insert-here
2000035A: B500  push { lr }
2000035C: F7FF  bl  20000340  --> 42+
2000035E: FFF0
20000360: BD00  pop { pc }
ok.
```

Warning

Inlining will fail if there are PC-relative BL opcodes in the definition to be inlined so carefully check the definition(s) in question using the disassembler before specifying INLINE flag or INLINE,

Now Inlining is used, note that “insert-here” now includes the code from “42+ ” rather than branching to it.

```
: 42+ 42 + inline ;
see 42+
20000422: B500  push { lr }
20000424: 362A  adds r6 #2A
20000426: BD00  pop { pc }

: insert-here 42+ ;
see insert-here
2000043A: B500  push { lr }
2000043C: 362A  adds r6 #2A
2000043E: BD00  pop { pc }
```

Note

Can Inline be used later within “insert-here” instead of in 42+ ?

No, this would only result in “insert-here” being Inlined if called by a later Word.

```
: 42+ 42 + ;
see 42+
2000044A: B500  push { lr }
2000044C: 362A  adds r6 #2A
2000044E: BD00  pop { pc }

: insert-here 42+ inline ;
see insert-here
20000462: B500  push { lr }
20000464: F7FF  bl  2000044A  --> 42+
20000466: FFF1
20000468: BD00  pop { pc }
```

See also

[Convert your Classic Kernel to RA]()

Smudge

Finishes a definition by flushing all flash buffers and inserts a link into the dictionary chain, making the new definition visible.

Vocabularies

July 2017, *great news!* [Manfred Mahlow’s]() Final Release of VOCs extension for Mecrisp-Stellaris 2.3.6-hook-find (vocs-0.6.2-FR) has arrived for testing. See the new [GPIO MODE Demo]()

Amforth

> In amforth there is a word that can place newly defined words in wordlists automatically. e.g. a word called foo:bar is placed as bar in wordlist foo. It is like a sibling of a dot (colon)-recognizer that can detect the foo:bar notation and searches bar in foo (only). This word does not affect the definition wordlist or the search order, not even temporarily. [mtrute]

lshift

Q: What does the LSHIFT Word do ?

Here are the Dictionary definitions of the *SHIFT* Words.

exactly ANS, some logical extensions

arshift	(x1 u - - x2)	Arithmetric right-shift of u bit-places
rshift	(x1 u - - x2)	Logical right-shift of u bit-places
lshift	(x1 u - - x2)	Logical left-shift of u bit-places

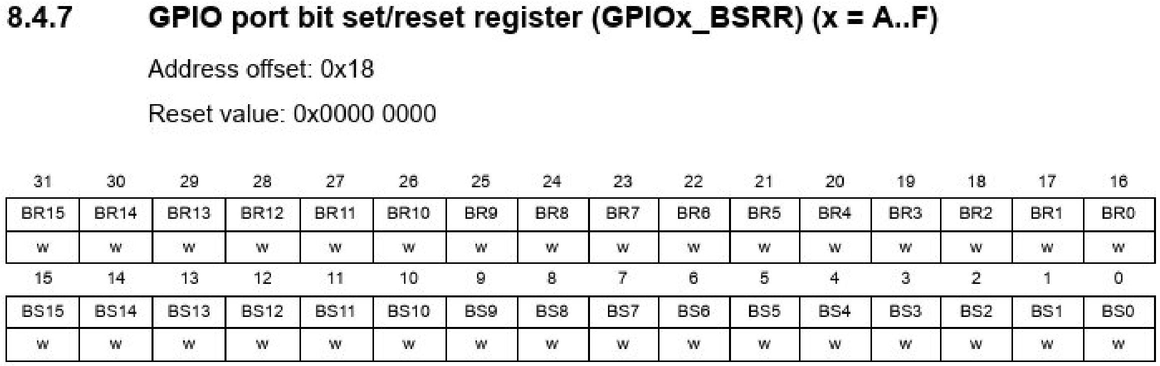
Example

```
: gpiob_bsrr_bs9   %1 9 LSHIFT gpiob_bsrr ! ;  \
gpiob_bsrr_bs9
```

GIOB BSRR Register

This is a 32 bit register and the picture below is taken from page 159 of the official STM **`STM32F0xx Reference Manual`**

Table 8.4.7



8.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A..F)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

These bits are write-only. A read to these bits returns the value 0x0000. Note: If both BSx and BRx are set, BSx has priority.

BIT	**0**	**1**
BS0	No Action	**SETS** BIT 0
BR0	No Action	**RESETS** BIT 0

How does it work ?

There is a “BSRR” register for every GPIO Port on the chip, and this example applies to GPIOB.

Every one of the 16 bits in GPIOB can be SET (1) or RESET (0) by storing a 1 in either its BS or BR Bitfield respectively.

Suppose we want to SET BIT 9 in GPIOB ? What we do is store a “1” to the BS9 position of the GIOB BSRR Register above.

There are many ways to do this, let’s look at some of them.

1) looking at the picture above we can store the binary value of %1000000000 into GIOB_BSRR. The “1” will only set BIT 9 and the “0“‘s will have no effect on the other bits.

```
%1000000000 GIOB_BSRR !
```

2) Binary %1000000000 is equal to decimal 512 so we could use that instead, exactly the same thing would happen, BIT 9 would be SET.

```
512 GIOB_BSRR !
```

3) Hexadecimal $200 is also equal to %1000000000, so we could use that to set BIT 9

```
$200  GIOB_BSRR !
```

Which one do you prefer ? The binary one does relate directly to the BIT in question, but it has a LOT of “0“‘s which are painful to type and painful to check. Decimal or Hex have the same effect but are mentally harder to equate to BIT 9 in Table 8.4.7. above.

Here is a alternative way to look at BIT 9 after it is SET and this is just a table of bits 0 to 31 showing bit 9 is set, and all the others are 0. lines 1 and 2 are the bit number and line 3 is the bit value.

```
3322222222221111111111
10987654321098765432109876543210
00000000000000000000001000000000
```

So back to the original question.

The LSHIFT word is a short way to specify BIT 9 and still maintain a easy mental relationship to Table 8.4.7. above. This is how it works.

The LSHIFT Word takes two parameters, the first parameter is the NUMBER TO BE SHIFTED, and the other is how many TIMES TO SHIFT THE NUMBER LEFT.

In our example. “1” is shifted 9 times left using the syntax below.

```
%1 9 lshift
```

Which results in the number 1

```
3322222222221111111111
10987654321098765432109876543210
00000000000000000000000000000001
```

being shifted into BIT POSITION 9 which is what we want.

```
3322222222221111111111
10987654321098765432109876543210
00000000000000000000001000000000
```

We then store that number into GIOB_BSRR like so, which in English reads “shift one, 9 spaces left then store the result into the GPIOB_BSRR register.

```
%1 9 lshift GPIOB_BSRR !
```

See also

Our example above is a template that was generated automatically, no one wrote it. See CMSIS-SVD for more details.

C Language also Left Shifts

Mecrisp-Stellaris Forth isnt the only programming language to left shift bits, look at the following include code by Keil. This code is to define GPIO_MODER_MODE0

```
#define GPIO_MODER_MODE0_Pos            (0U)
#define GPIO_MODER_MODE0_Msk            (0x3UL <<
GPIO_MODER_MODE0_Pos)        /*!< 0x00000003 */
#define GPIO_MODER_MODE0                GPIO_MODER_MODE0_Msk
```

I do it this way in Forth

```
: GPIOC_MODER_MODER0   ( %XX -- ) 0 lshift GPIOC_MODER bis! ;
\ GPIOC_MODER_MODER0

%01 constant OUTPUT

OUTPUT GPIOC_MODER_MODER0             \ Configure GPIOC_0 as
a OUTPUT
```

Loop Unrolling

Loop unrolling, also known as loop unwinding, is a loop transformation technique that attempts to optimize a program’s execution speed at the expense of its binary size

Postpone

(My thanks to rdrop-exit on irc.freenode #forth for helping me understand this.)

postponing has the effect of delaying the normal compile-time action of a word, but what that action normally is depends on whether the word being postponed is immediate or not

Example

```
: THINKING ( -- ) ." I'm deciding... " ;
: NEWIF ( x -- )  postpone THINKING  postpone IF ; immediate
: FOO ( x -- ) newif ." yes" else ." no" then ;

1 FOO I'm deciding... yes ok.
0 FOO I'm deciding... no ok
```

Note

THINKING and FOO are NON-IMMEDIATE Words, IF and NEWIF are IMMEDIATE Words. See Dictionary Flags

```
Flags: 00000630 Name: if
Flags: 00000000 Name: FOO
Flags: 00000010 Name: NEWIF
Flags: 00000000 Name: THINKING
```

FLAG Codes

Taken from mecrisp-stellaris-2.4.7/mecrisp-stellaris-source/common/datastackandmacros.s

Flag	Description
0x0000	Visible, non-immediate
0x0010	Immediate
0x0030	Immediate+ Inline means: Immediate, Compile-Only

Registerliteral,

This is used to generate RELATIVE machine code to use when INLINING.

Registerliteral, generates the shortest possible sequence to get x into given low Register using movs adds lsls sequences. 1. M0: A movs-lsls-adds… sequence 2. M3/M4: movs / movs-mvns / movw / movw-movt

This is used instead of the command “ldr=” which generates ABSOLUTE addressing. A movs adds lsls sequences needs to be used instead to load a literal in a register.

Example

```
: rlit
  [
  1000 0 registerliteral,
  ]
;

see rlit
20000654: B500  push { lr }
20000656: 20FA  movs r0 #FA
20000658: 0080  lsls r0 r0 #2
2000065A: BD00  pop { pc }
```

HTML

Warning

Under Construction

Low Power on STM32F0

One Lone Electronics Technician battles with STM32F0 Low Power Modes

Documentation

If you’re a Forth addict like me, you know that mastering unfamiliar hardware involves reading what documentation you can find, understanding it, then trying out your assumptions based on the new knowledge.

In this case, the ST documentation was incomplete, ambiguous, maybe outdated, and I certainly misunderstood some of it.

Note

Caveat, I prefer ST Documentation, it’s not perfect, but I think it’s nicely laid out and easy to read. ST had a *massive* job documenting Cortex M plus their peripherals, registers, bitfields etc.

Documentation Fragmentation

This was my main issue, because the key (SCR_SLEEPDEEP) to initiating ‘Standby Mode’ wasn’t obvious to me when I read the STM32F0 Technical and Programming pdfs.

For one thing, I rely heavily on CMSIS-SVD to create all the STM32F051 Memory Mapped Words and Bitfields for my code and while CMSIS-SVD has all the MCU Peripheral data I need, there is no mention of the System Control Register.

The System Control Register information was hiding in STM document PM0215 on page 81. https://www.st.com/resource/en/programming_manual/dm00051352.pdf

Researching OnLine Code Examples

Forth

To put it bluntly, there aren't any STM32F0 low power code examples for Forth online that I can find.

```
Google: "forth stm32f0 low power"
No results found for "forth stm32f0 low power".
```

C Code Examples

These are everywhere, but usually not much use to me.

<rant>: Why I Don't Use C Programming Language Examples

STM32F0 Low Power

One can't just consider the Low Power Modes by themselves, because they fit in the middle of the whole process.

Something activates the Low Power process, and something wakes it up. Both the activate and wakeup process can be somewhat complex in themselves, involving the wide range of peripheral devices and interrupts.

Activating Low Power

I'll try to keep it simple, and start the Low Power Modes using the WFI instruction on a STM32F0 Discovery Board.

WFI (Wait For Interrupt)

Is a hint instruction that suspends execution until one of the following events occurs:

```
An exception
An interrupt becomes pending which would preempt if PRIMASK was clear
A Debug Entry request, regardless of whether Debug is enabled.
```

Note

WFI is intended for power saving only. When writing software assume that WFI might behave as a NOP operation.

WFI Forth Word

Presented below as WFI is not yet in the Mecrisp-Stellaris Dictionary.

```
: wfi ( -- mcu asleep ) [ $BF30 h, ] inline ;  \ WFI Opcode, Wait For Interrupt, 
enters
```

Low Power Register/Bitfields

There are three Bitfields (SLEEPDEEP, PDDS and LPDS) that control the low power modes.

Bitfield Name	Description	Register	Memory	Bit
SLEEPDEEP	System Control Register	SCR	0xE000ED10	2
PDDS	Power down deep sleep	PWR_CR	0x40007000	1
LPDS	Low power deep sleep	PWR_CR	0x40007000	2

Low Power Modes

```
X = Don't Care
All Low Power Modes are initiated by the WFI or WFE instruction
```

Current	Voltage	Clock MHz	Mode	SLEEPDEEP	PDDS	PDS	REGULATOR	CALL WFI?
16 mA	3.0	48	RUN	X	X	X	ON	NO
3.1mA	3.0	8	RUN	X	X	X	ON	NO
1.4 mA	3.0	8	SLEEP	0	X	X	ON	YES
0.9 mA	3.0	8	STOP	1	0	0	ON	YES
0.9 mA	3.0	8	STOP LOW POWER	1	1	0	LOW POWER	YES
4.8 uA	3.0	8	STANDBY	1	X	1	OFF	YES

Internal 1.8 Volt Regulator

The voltage regulator is always enabled after Reset. It works in five different modes.

Mode	Description
Run	The regulator supplies power to the core, memories and digital peripherals (everywhere).

Mode	Description
Sleep	The CPU clock is turned off, all other clocks remain running.
Stop	All clocks are stopped, the regulator supplies power only to the registers and SRAM
Stop Low Power	All clocks are stopped except when required by any peripheral that can work in STOP mode. Registers and SRAM have power.
Standby	The regulator is powered off except for the Standby circuitry and the RTC. The contents of the registers and SRAM are lost.

Note

One ongoing mystery is which peripherals “can work in STOP mode”, and which ones cannot ? more testing is needed.

Wakeup the Device

Mode	Wakeup Methods
Sleep	Any interrupt can wake-up the device.
Stop	The User Pushbutton via A WKUP pin
Stop Low Power	The User Pushbutton via A WKUP pin, RTC
Standby	The User Pushbutton via A WKUP pin (rising edge), IWDG reset, external reset on NRST pin, RTC alarm.

My Example Forth Code

`Download: f0-lowpower-1.fs`

```
\ Program Name: f0-lowpower-1.fs
\ This program may require other support files listed in preload.sh which need to be
loaded first
\ Date: Mon 12 Nov 2018 11:43:22 AEDT
\ Copyright 2018  t.porter <terry@tjporter.com.au>, licensed under the GPL
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F051, Board: STM32F0 Discovery Board
\ Clock: 8 Mhz using the internal STM32F051 RC clock, unless otherwise stated
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\
\ Purpose: To test the STM32F051 LOW Power Modes
\
\ ---------------------------------------------------------------------------\
 compiletoram

  \ SCR System Control Register, see PM0215 page 81 and README-f0-lowpower.txt
   $E000ED10  CONSTANT  SCR      ( System Control )   \ read/write, Reset value
$00000000


   \ SCR () $00000000
   \ SYSTEM CONTROL REGISTER                                   S S
   \                                                       S   L E
   \ controls features of entry to and exit from           E   E E
   \ LOW POWER state.                                      V   E P
   \                                                       O   P O
   \                                                       N     N
   \                                                       P   D E
   \                                                       E   E X
   \                                                       N   E I
   \                                                       D   P T
   \ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~| 4| ~| 2| 1| ~
   \ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

   : sys-init
   ( RCC_APB1ENR_PWREN-SET )  %1  28  lshift $4002101C  bis!  \ Enable APB1 peripheral
bus
   ;

   : green-on ( -- )  %1   9  lshift $48000818 bis! ;  \ GPIOC_BS9 Green LED ON
   : green-off ( -- ) %1  25  lshift $48000818 bis! ;  \ GPIOC_BR9 Green LED OFF
   : wfi ( -- mcu asleep ) [ $BF30 h, ] inline ;  \ WFI Opcode, Wait For Interrupt
   : SLEEPDEEP=0 1 2 lshift SCR bic! ;       \  SLEEPDEEP bit 2; 0 = sleep, 1 = deep
sleep
   : SLEEPDEEP=1 1 2 lshift SCR bis! ;
   : PDDS=0 %1  1  lshift $40007000  bic! ;  \ PWR_CR_PDDS; Power down SLEEPDEEP
   : PDDS=1 %1  1  lshift $40007000  bis! ;
   : LPDS=0 %1  1  lshift $40007000  bic! ;  \ PWR_CR_LPDS; Low-power deep sleep
   : LPDS=1 %1  1  lshift $40007000  bis! ;


   : wakeup-config
   ( PWR_CR_CWUF-SET )   %1  2  lshift $40007000  bis!   \ PWR_CR_CWUF; Clear wakeup
flag
   ( PWR_CSR_EWUP-SET )  %1  8  lshift $40007004  bis!  \ PWR_CSR_EWUP; Enable WKUP
pin
   ;

   : userpb-interrupt-config
   ( SYSCFG_EXTICR1_EXTI0-SET ) ( %0000 -- ) %1111  0  lshift $40010008  bic!  \ x000:
PA[x] pin GPIO-A Pin 0  Pin trigger
   ( NVIC_ISER_SETENA-SET )  %100000    0  lshift $E000E100  bis!  \ EXTI0_1  EXTI
```

```
Line[1:0] interrupts,  it's connected to PA0 1
   ( EXTI_IMR_MR0-SET )  %1  0  lshift $40010400  bis!  \ EXTI_IMR_MR0; Interrupt
request on line 0 enabled
   ( EXTI_FTSR_TR0-SET )  %1  0  lshift $4001040C  bis!  \ EXTI_FTSR_TR0; Falling
trigger event line 0 when USR pbbutton is released
   ( EXTI_EMR_MR0-SET )  %1  0  lshift $40010404  bis!  \ EXTI_EMR_MR0; Event request
on line 0 enabled
   ;

   : disable-pb-int ( EXTI_IMR_MR0-RESET )  %1  0  lshift $40010400  bic! ; \
EXTI_IMR_MR0; Interrupt request on line 0 disabled

   : pb-interrupt-handler
   ( EXTI_PR_PR0-SET )  %1  0  lshift $40010414  bis!  \ EXTI_PR_PR0; Pending bit 0,
clear
   100 0 do nop loop             \ delay before sending message after restart
   cr ." System awake ... " cr
   green-on
   disable-pb-int
   ;

   : pb-interrupt-hook
   ['] pb-interrupt-handler irq-exti0_1 ! \ tie the interrupt handler to the interrupt
   ;


\ All modes, many are redundant, left in for completeness, try them!

   : SLEEP cr   \ 1.7 mA  Repeat without reboot.
    ." Entering SLEEP MODE: CPU clock off, all peripherals are running " cr
    ." Press the blue USER push button to wake up ..... " cr
      userpb-interrupt-config
      SLEEPDEEP=0
      PDDS=0
      LPDS=0
      green-off
      wfi
   ;


   : SLEEP-1 cr  ( redundant) \ 1.7 mA  Repeat without reboot.
    ." Entering SLEEP MODE: CPU clock off, all peripherals are running " cr
    ." Press the blue USER push button to wake up ..... " cr
      userpb-interrupt-config
      SLEEPDEEP=0
      PDDS=0
      LPDS=1
      green-off
      wfi
   ;

   : SLEEP-2 cr ( redundant) \ 1.7 mA  Repeat without reboot.
    ." Entering SLEEP MODE: CPU clock off, all peripherals are running " cr
    ." Press the blue USER push button to wake up ..... " cr
      userpb-interrupt-config
      SLEEPDEEP=0
      PDDS=1
      LPDS=0
      green-off
      wfi
   ;

   : SLEEP-3 cr ( redundant) \ 1.7 mA Repeat without reboot.
    ." Entering SLEEP MODE: CPU clock off, all peripherals are running " cr
    ." Press the blue USER push button to wake up ..... " cr
      userpb-interrupt-config
      SLEEPDEEP=0
      PDDS=1
      LPDS=1
      green-off
      wfi
   ;


   : STOP cr   \ 0.802 mA  Repeat without reboot
   ." Press the blue USER push button to wake up ..... " cr
   ." All clocks are stopped, the regulator supplies power only to the registers and
SRAM " cr
      userpb-interrupt-config
      wakeup-config
      SLEEPDEEP=1
      PDDS=0
      LPDS=0
      green-off
      wfi
   ;

    : STOP-LOW-POWER cr      \ 0.802 mA   Repeat without reboot.
   ." All clocks are stopped except when required by any peripheral that can work in
STOP mode. Registers and SRAM have power. " cr
   ." Press the blue USER push button to wake up " cr
      userpb-interrupt-config
      wakeup-config
      SLEEPDEEP=1
      PDDS=1
      LPDS=0
      green-off
      wfi
   ;

   : STANDBY cr      \ 4 uA . Reboots
   ." Press the blue USER push button to wake up " cr
   ." This will REBOOT the MCU, all RAM will be lost " cr
      userpb-interrupt-config
      wakeup-config
      SLEEPDEEP=1
      PDDS=0
      LPDS=1
      green-off
      wfi
   ;

   : STANDBY-1 cr ( redundant)  \ 4 uA . Reboots
   ." Press the blue USER push button to wake up ..... " cr
      userpb-interrupt-config
      wakeup-config
      SLEEPDEEP=1
      PDDS=1
      LPDS=1
      green-off
      wfi
   ;


   : main                       \ my multitasking system, harmless to leave it in
     pause
   ;
```

```
: init-general
 sys-init
 pb-interrupt-hook
 green-on
;


\ init-general                \ uncomment to run on your system
```

Ihex 32 Bit

Forth Project Cloning

When projects are built using the non-interactive C Programming Language, the Source Code is compiled to Machine Code on a PC. This leaves a single binary file on the PC ready to flash to more MCU’s if required.

When projects are built using interactive Mecrisp-Stellaris Forth, the Source Code is compiled to Machine Code on the MCU itself. The only compiled Machine Code resides in the MCU “Dictionary”.

To create multiple identical MCU’s with Mecrisp-Stellaris Forth there are two choices.

1. Upload all the Forth source files again to additional MCU(s). This means there is no binary file record of the project.
2. Use iHex.fs below to produce a single, bootable Machine Code binary file on a PC ready to flash to more MCU’s if required. This also allows easy redistribution of the Forth project and is the method used for all binaries distributed via this site.

iHex.fs

iHex.fs dumps the complete Dictionary binary content when the target is cloned.

This method uses “arm-none-eabi-objcopy” to then convert the ASCII data back into a bootable binary image file on your PC so it may be flashed to another MCU thereby “cloning” the target chip.

Intel Hex

The “Intel Hex” (iHex) format is used to transfer binary data over ASCII mediums, such as via a serial terminal. The data is also checksummed for later verification.

Ihex 04 Type Extended Addresses

Intel Hex is a old format from the 8 bit MCU days, when 16 addressing was common. Nowadays MCU’s have much larger address with 32 bit addressing used in ARM Cortex-M.

Extended Addressing is used to extend the capability of Intel Hex from a maximum of 64kB to 4GB. It does this by inserting ‘extended addresses’ into the iHex file every 64kB as shown below.

Ihex 04 Type Extended Addresses Example

Showing type 04 Type Ihex Extended Addresses. These are calculated in ihex.fs up to 4GB.

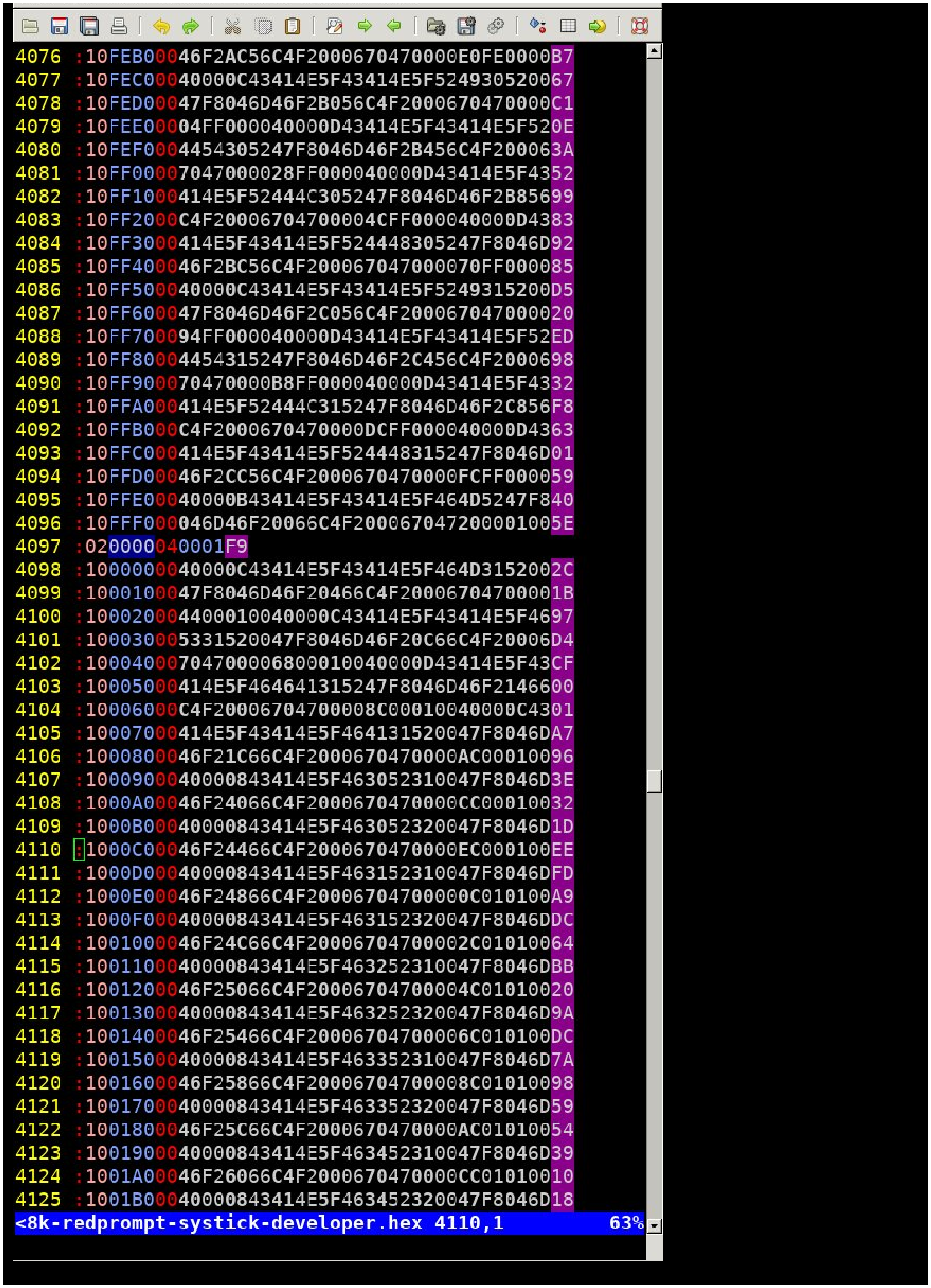
Hex	Extended Address	Bytes
$10000	:020000040001F9	64 kB
$20000	:020000040002F8	128 kB
$30000	:020000040003F7	192 kB
$40000	:020000040004F6	256 kB
$50000	:020000040005F5	320 kB
$60000	:020000040006F4	384 kB
$70000	:020000040007F3	448 kB
$80000	:020000040008F2	512 kB

Hex	Extended Address	Bytes
$90000	:020000040009F1	576 kB
$A0000	:02000004000AF0	640 kB
$B0000	:02000004000BEF	720 kB
$C0000	:02000004000CEE	786 kB
$D0000	:02000004000DED	851 kB
$E0000	:02000004000EEC	917 kB
$F0000	:02000004000FEB	983 kB
$100000	:020000040010EA	1 MB

Ihex Dump Example

Showing first extended address insertion “:020000040001F9” at $10000 - 64kB

File
Edit
Tools
Syntax
Buffers
Window
Help



```
:10FEB00046F2AC56C4F2000670470000E0FE0000B7
:10FEC00040000C43414E5F43414E5F524930520067
:10FED00047F8046D46F2B056C4F2000670470000C1
:10FEE00004FF000040000D43414E5F43414E5F520E
:10FEF0004454305247F8046D46F2B456C4F200063A
:10FF00007047000028FF000040000D43414E5F4352
:10FF1000414E5F52444C305247F8046D46F2B85699
:10FF2000C4F2000670470004CFF000040000D4383
:10FF3000414E5F43414E5F524448305247F8046D92
:10FF400046F2BC56C4F20006704700070FF000085
:10FF500040000C43414E5F43414E5F5249315200D5
:10FF600047F8046D46F2C056C4F200067047000020
:10FF700094FF000040000D43414E5F43414E5F52ED
:10FF80004454315247F8046D46F2C456C4F2000698
:10FF900070470000B8FF000040000D43414E5F4332
:10FFA000414E5F52444C315247F8046D46F2C856F8
:10FFB000C4F2000670470000DCFF000040000D4363
:10FFC000414E5F43414E5F524448315247F8046D01
:10FFD00046F2CC56C4F2000670470000FCFF000059
:10FFE00040000B43414E5F43414E5F464D5247F840
:10FFF000046D46F20066C4F2000670472000010005E
:020000040001F9
:1000000040000C43414E5F43414E5F464D3152002C
:1000100047F8046D46F20466C4F20006704700001B
:10002000440001004000C43414E5F43414E5F4697
:100030005331520047F8046D46F20C66C4F20006D4
:1000400070470000680001004000D43414E5F43CF
:10005000414E5F464641315247F8046D46F2146600
:10006000C4F20006704700008C000100400000C4301
:10007000414E5F43414E5F464131520047F8046DA7
:1000800046F21C66C4F2000670470000AC00010096
:100090004000843414E5F463052310047F8046D3E
:1000A00046F24066C4F2000670470000CC00010032
:1000B0004000843414E5F463052320047F8046D1D
:1000C00046F24466C4F2000670470000EC000100EE
:1000D0004000843414E5F463152310047F8046DFD
:1000E00046F24866C4F200067047000000C010100A9
:1000F0004000843414E5F463152320047F8046DDC
:1001000046F24C66C4F20006704700002C01010064
:100110004000843414E5F463252310047F8046DBB
:1001200046F25066C4F20006704700004C01010020
:100130004000843414E5F463252320047F8046D9A
:1001400046F25466C4F20006704700006C010100DC
:100150004000843414E5F463352310047F8046D7A
:1001600046F25866C4F20006704700008C01010098
:100170004000843414E5F463352320047F8046D59
:1001800046F25C66C4F2000670470000AC01010054
:100190004000843414E5F463452310047F8046D39
:1001A00046F26066C4F2000670470000CC01010010
:1001B0004000843414E5F463452320047F8046D18
<8k-redprompt-systick-developer.hex 4110,1                63%
```

Note

This page replaces older versions cloning and cloning-integrated

Description

Ihex.fs dumps the entire Dictionary contents to the terminal in Intel Hex Extended Ascii format, and can handle sizes up to the maximum 4GB available with 32 bit addressing as used by Cortex-M.

ihex.fs reads the STM32 MCU “Flash Size Register” to determine the maximum size and won’t attempt to read beyond that. This is to prevent a ‘exception’ being thrown by the MCU.

This works fine except where STM have actually included more Flash than stated in the “Flash Size Register”, such as in the STM32F103C8 which declares 64kB but usually has 128kB of Flash. This situation is handled with a parameter option to “clone”, see below.

Note

The “Flash Size Register” address is *DIFFERENT BETWEEN DIFFERENT MODELS* and must be changed to suit *YOUR* MCU. The address is given in the STM Technical Reference Manual and a few examples below.

Operation

- Load ihex.fs
- Enable your terminal log to capture the output, and then run the “clone” program as shown below. “Clone” requires TWO parameters, the first is a “flash-size-register-address” modifier, the second is the “flash-size-register-address” itself. This may be in Hex or Decimal as desired.
- There is *NO PARAMETER CHECKING*. If you make a mistake the program will probably generate a Exception and/or produce incorrect results.

  Usage: “1| 2 flash-size-register-address clone”

  ‘1’ = normal use

  ‘2’ = only for a STM32F103C8 which reports 64kB of Flash but actually has double (128kB) OR if you suspect your chip MAY have double the flash advertized. Note: will crash this

```
     program with a Exception if it
     doesn’t!

     ‘flash-size-register-address’ =
     “Flash size data register address”.
     Check your STM reference manual
     “Device electronic signature”
     section.

     ‘clone’ name of the Forth program as
     listed below in the source

     Example usage for:

     STM32F051: “1 $1FFFF7CC clone”

     STM32F103CB: “1 $1FFFF7E0 clone”

     STM32F103C8: “2 $1FFFF7E0 clone”

The iHex dump starts with the word “clone”
and ends with the word “clone_end” for easy
parsing to remove any extraneous text
before using ‘arm-none-eabi-objcopy’.

Once you have the clean iHex File process
it as below. Note that any junk,extra text
or checksum errors in this file will cause
arm-none-eabi-objcopy to abort with a error
message.

” arm-none-eabi-objcopy -I ihex -O binary
<your-iHex-logfile> <binaryfile-name.bin> “
```

- After the dump is complete, save/disable your terminal log and edit-out the pre-amble (everything before and including “clone”) and post-amble (remove “clone_end” at the very end of the file) text.

Note

Arm-none-eabi is required but you should have it installed if you have compiled Mecrisp-Stellaris previously.

Sourcecode

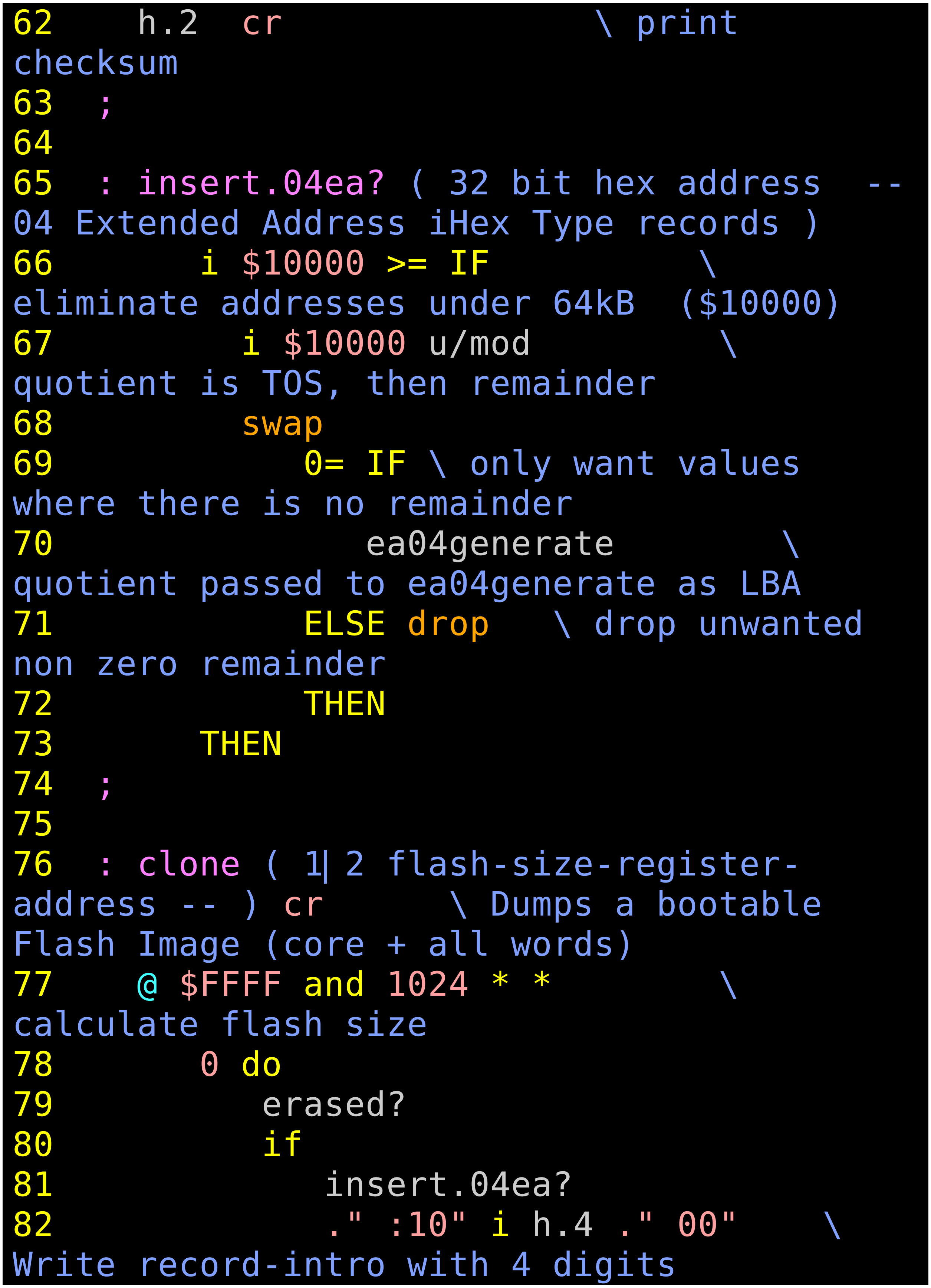
Download: ihex.fs

```
\ Program Name: ihex.fs
\ Date: Sun 5 Jan 2020 21:19:51 AEDT
\ Copyright 2020 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV2
\ For Mecrsp-Stellars by Matthas Koch.
\ https://sourceforge.net/projects/mecrisp/
\ MCU: STM32Fxxx
\ This Forth Program : Dumps a iHex image of the target MCU dictionary to the
\ terminal where it can be captured in a terminal log and turned into a
\ bootable binary file clone of the
```

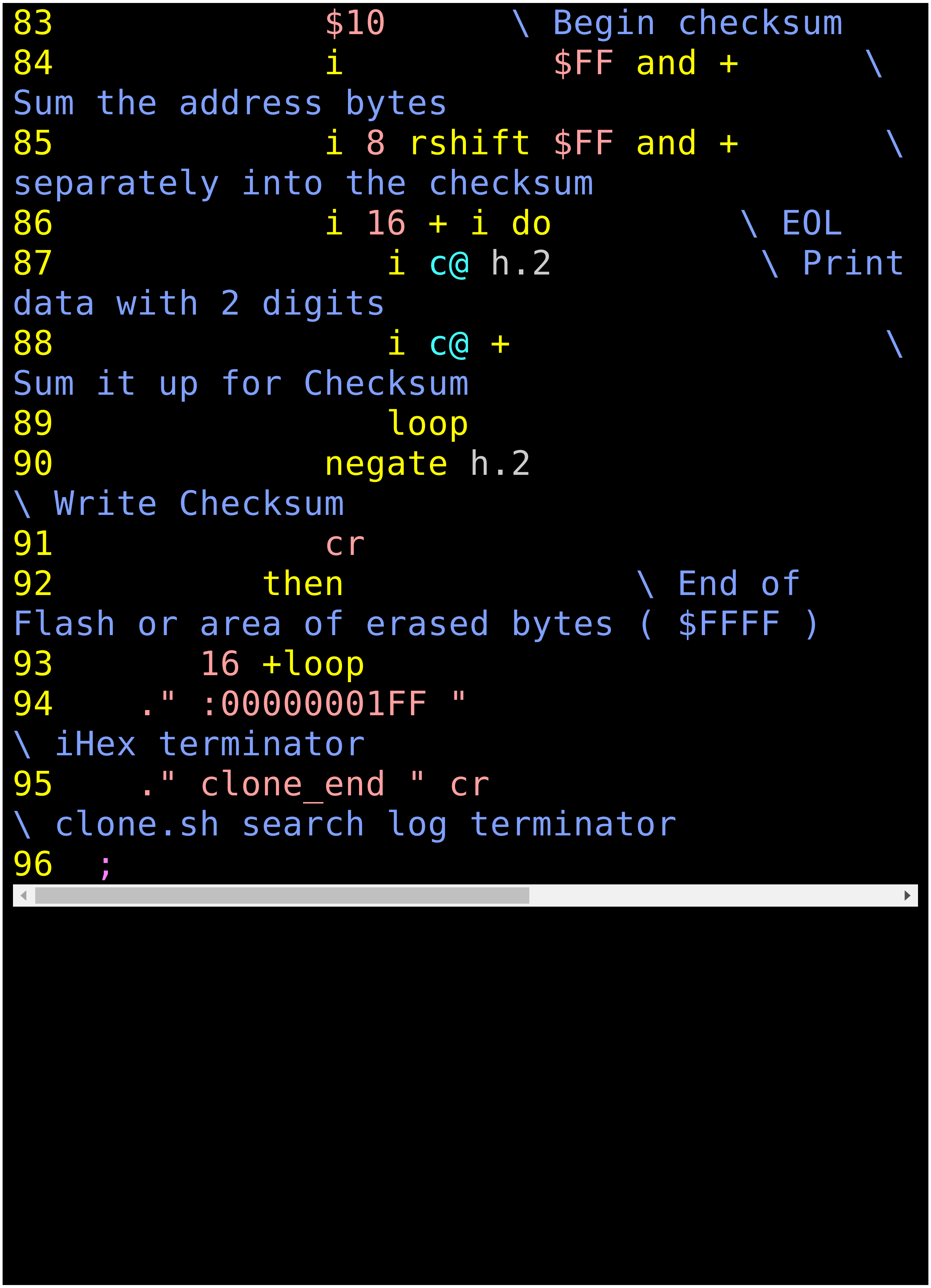
```
target.
\ ---------------------------------------------------------------------------------------\
\
\     Usage:    "1|2 flash-size-register-address clone"
\
\     '1' = normal use
\
\     '2' = only for a STM32F103C8 which reports 64kB of Flash but actually has double (128kB)
\     OR if you suspect your chip MAY have double the flash advertized. Note: will crash
\     this program with a Exception if it doesn't!
\
\     'flash-size-register-address' = "Flash size data register address". Check your STM
\     reference manual "Device electronic signature" section.
\
\     'clone' name of the program as listed below in the source
\
\     Example usage for:
\     STM32F051:     "1 $1FFFF7CC clone"
\     STM32F103CB:   "1 $1FFFF7E0 clone"
\     STM32F103C8:   "2 $1FFFF7E0 clone"
```

```
\
\ The iHex dump starts with the word "clone" and ends with the word "clone_end" for
\ easy parsing to remove any extraneous text before using 'arm-none-eabi-objcopy'.
\
\ Once you have the iHex File process it as below.
\
\ " arm-none-eabi-objcopy -I ihex -O binary <your-iHex-logfile> <binaryfile-name.bin> "
\ -------------------------------------------------------------------------------------\
\ Notes:
\ Can produce extended iHex files up to 32 bits, i.e. 4 GB.
\ iHex Extended Addressing type 04 is used
\ ihex.fs reads the STM32 MCU "Flash Size Register" to determine the maximum size and won't
\ attempt to read beyond that because a Exception may be thrown if it is the Flash end.
\ Check your STM reference manual for your MCU's "Flash Size Register" address.
\
\ flash-size-register-address    MCU
\ ---------------------------    -------
```

```
---
\ $1FFFF7CC STM32F0XX
\ $1FFFF7E0 STM32F1XX
\ $1FFF7A22 STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439
\ -------------------- Shouldn't need to change anything below ------------------------- \

 : erased? ( -- )    \ Check Flash bytes NOT erased.
   0                  \ Flag of 0 = erased byte, ignore data afterwards
   i 16 + i do         \ Scan data
      i c@ $FF <> or    \ Set flag if there is a non-$FF byte. Some chips have Flash = $00 as erased.
   loop
 ;

 : ea04generate ( -- print 04 Extended Address Type record )
   ." :02000004"              \ Write 04 Extended Address Type record preamble
   dup h.4                     \ quotient = LBA
   $06 + not $FF and 1 +        \ calculate checksum. $06 is record preamble value.
```



```
  h.2  cr                \ print checksum
;

: insert.04ea? ( 32 bit hex address  -- 04 Extended Address iHex Type records )
     i $10000 >= IF          \ eliminate addresses under 64kB  ($10000)
       i $10000 u/mod          \ quotient is TOS, then remainder
       swap
          0= IF \ only want values where there is no remainder
             ea04generate        \ quotient passed to ea04generate as LBA
          ELSE drop   \ drop unwanted non zero remainder
          THEN
     THEN
;

: clone ( 1| 2 flash-size-register-address -- ) cr      \ Dumps a bootable Flash Image (core + all words)
  @ $FFFF and 1024 * *        \ calculate flash size
     0 do
        erased?
        if
           insert.04ea?
           ." :10" i h.4 ." 00"    \ Write record-intro with 4 digits
```



```
            $10      \ Begin checksum
            i          $FF and +      \ Sum the address bytes
            i 8 rshift $FF and +       \ separately into the checksum
            i 16 + i do         \ EOL
               i c@ h.2          \ Print data with 2 digits
               i c@ +                  \ Sum it up for Checksum
               loop
            negate h.2                 \ Write Checksum
            cr
         then                 \ End of Flash or area of erased bytes ( $FFFF )
      16 +loop
   ." :00000001FF "                    \ iHex terminator
   ." clone_end " cr                   \ clone.sh search log terminator
 ;
```

Loadmeter

During embedded development, an accurate idea of the MCU load is important, and it’s nice to have it confirmed by a second source … enter the “Loadmeter”.

A 53 year old Sangamo Weston uA meter

MADE IN ENGLAND
SW-9418687-64
dB
RESIS 700 Ω
F.S.=50μA

See also

This project was featured on Hackaday: https://hackaday.com/2017/05/13/microcontroller-load-meter-tells-you-how-hard-its-currently-working/

Cooperative Multitasking

A cooperatively multitasked system relies on each process regularly giving up time to other processes on the system. One poorly designed *task* or *process* can consume all of the CPU time for itself, either by performing extensive calculations or by *busy waiting*; both would cause the whole system to *hang*.

Whilst a *cooperative scheduler* starts each *task*, they must voluntarily give back control when they are finished or even during stages of their operation by using the *pause* Word.

Note

busy-waiting, busy-looping or spinning is a technique in which a process repeatedly checks to see if a condition is true

How Does It Work ?

A *loadmeter-task* sends a short pulse to a Microamp Meter. Any other tasks delay the rate at which *loadmeter-task* can send pulses, therby reducing the *duty cycle* of the square wave pulse train to the meter, giving a *lower* reading and indicating more *load*.

A moving coil meter is ideal because the mechanical inertia will average the duty cycle of the pulse train.

Calibration

The Loadmeter is calibrated with a trimpot for a point near the high reading of the meter, (in this case the “0” on the DB meter scale) to indicate the minimum load of the default system. This is done while *only* the *loadmeter-task* is running.

Usage

Once the *loadmeter-task* is running, subsequent *tasks* will cause the Loadmeter pointer to move to the left, indicating they are using MCU time, which is a increase in *load*.

While indicating MCU load, a steady pointer shows ideal rapid cycling between *tasks*, but a pulsing Loadmeter pointer indicates a *task* may be *hogging* the MCU or perhaps the “pause” Word is in the wrong place, such as at the end of a 1 second delay, when it should be in the loop counter instead ?

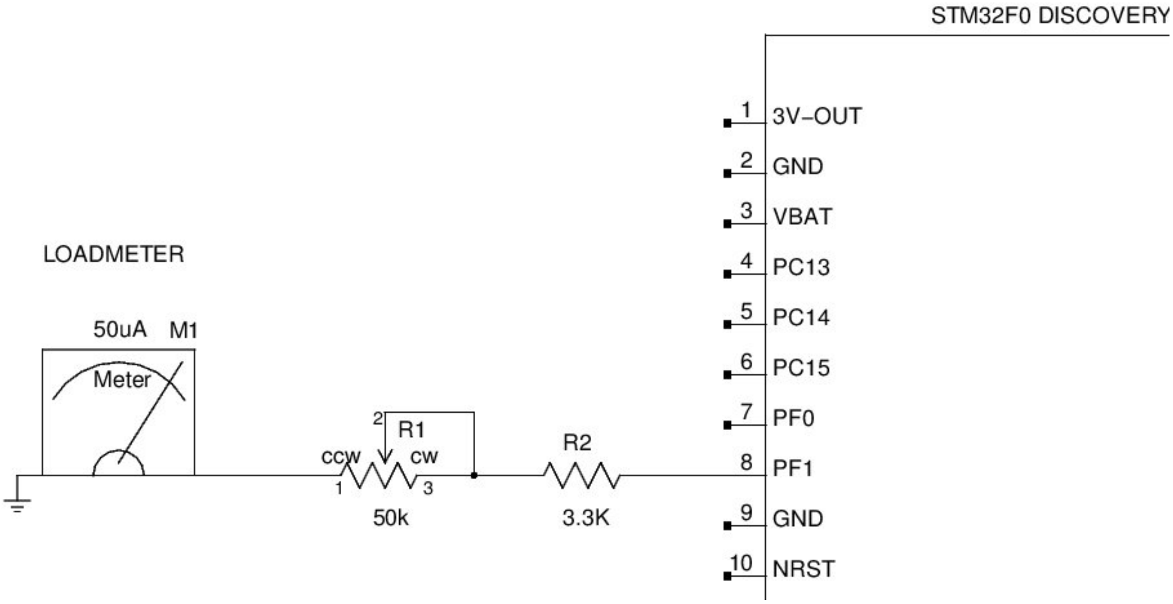
The Loadmeter also indicates:

1. How additional “pause” Words can have zero effect
2. Whether a *task* can be made more efficient with better code or perhaps by using a timer for a delay instead of spinning ?

Schematic Diagram

A STM32F0 Discovery board was used.

Loadmeter Schematic Diagram


STM32F0 DISCOVERY
1 3V–OUT
2 GND
3 VBAT
4 PC13
5 PC14
6 PC15
7 PF0
8 PF1
9 GND
10 NRST
LOADMETER
50uA M1
Meter
2 R1
CCW CW
1 3
50k
R2
3.3K

Source Code and Notes

Notes:

- The F0 Discovery board lacks RAM for more tasks, having only 8KB of RAM. Mecrisp-Stellaris will raise a 'Not Enough Ram' error if one more task is added.
- This is all running from Flash so I can fit in the 5 tasks, if ms-loadmeter.fs is run from ram, then a task must be removed.
- The USER push button lights the GREEN LED when depressed.
- The Blinky flashes the BLUE LED on and off about once a second
- Delays 1 and 2 just delay for a second each, they are there to waste time.
- All ancillary files are available from the library
- **`ms-loadmeter.fs`** and **`ms-loadmeter.memmap.fs`** are available here.

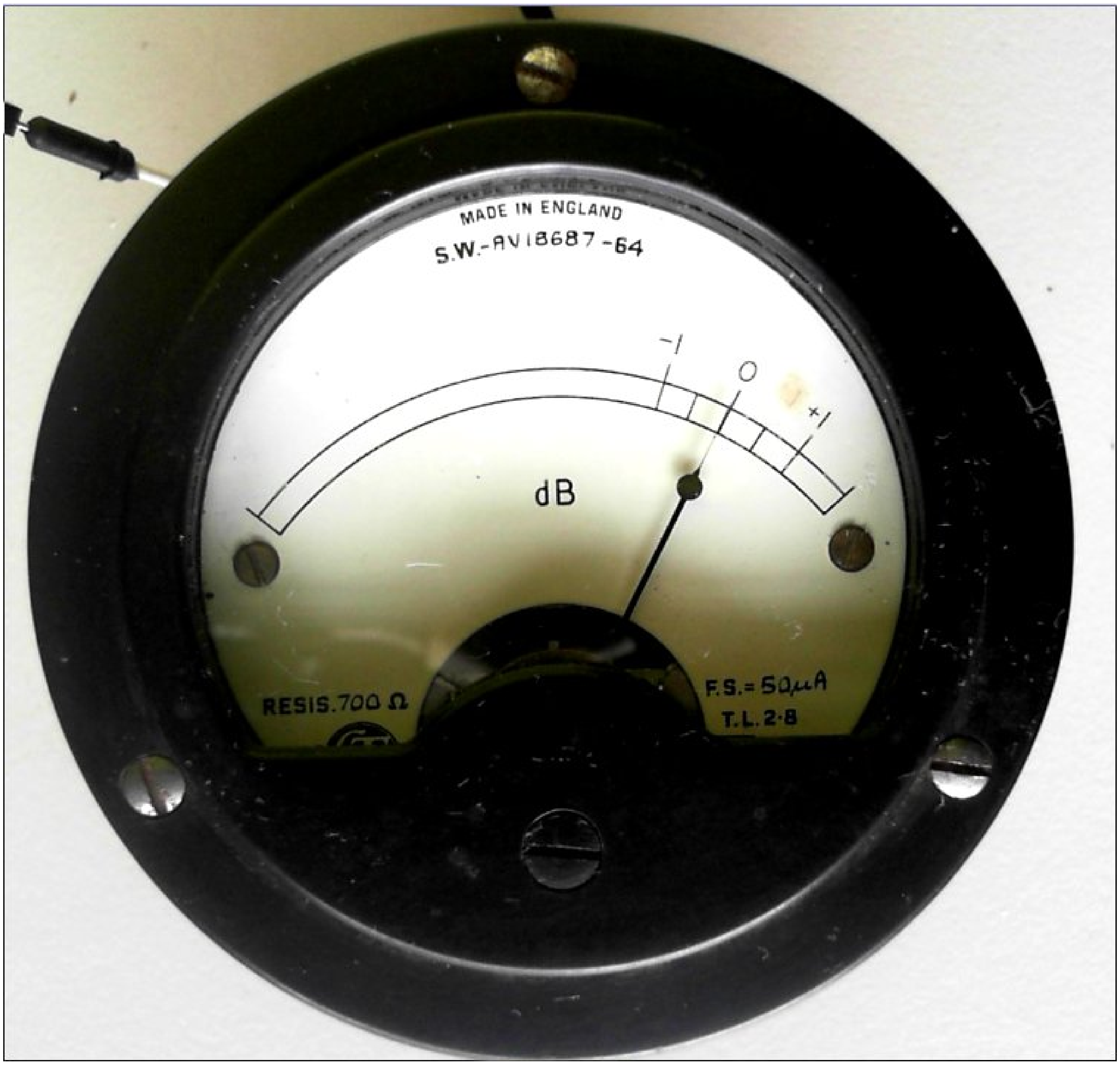
```
\ Program Name: ms-loadmeter.fs
\ Edit template.xml, commenting out unwanted register lines (
<!- commented out -> F9 ), then run 'make'
\ Date: Tue  9 May 2017 05:29:17 AEST
\ Copyright 2017  t.porter <terry@tjporter.com.au> Released
under the GPL
\ For Mecrisp-Stellaris by Matthias Koch
\ Chip: STM32F051
\ Board: STM32F0 Discovery Board
\ Terminal: e4thcom Copyright (C) 2013-2017 Manfred Mahlow;
https://wiki.forth-ev.de/doku.php/en:projects:e4thcom
\ Clock: 8 Mhz using the internal STM32F051 RC clock, unless
otherwise stated
\ All register names must be CMSIS-SVD compliant
\
\ This Program Does: Indicates MCU load using a external uA
moving coil meter.
\
\ Inputs: none
\ Outputs: PF-1
\
\ ------------------ USER CODE HERE ------------------ \

compiletoflash                      \ seems that
compiletoflash is reset to compiletoram after each upload ?
#require f0-id.fs                   \ Optional development
word

compiletoflash
#require memstat.fs                 \ Optional development
word

compiletoflash
#require dict.fs                    \ Optional development
word

compiletoflash
#require f0-legends.fs

compiletoflash
#require ms-loadmeter.memmap.fs

compiletoflash
#require multitask.fs

compiletoflash
```

```
\ Initialisations
%01  2 lshift GPIOF_MODER bis!      \ use PF-1 as meter
driver o/p
%01  18 lshift GPIOC_MODER bis!     \ PC-9 as output
%01  16 lshift GPIOC_MODER bis!     \ PC-8 as outpu


: .5s-delay 25000 0 do pause loop ;                 \ Used by
Blinky
: delay-1 50000 0 do pause loop ;                   \ 1
second delay
: delay-2 50000 0 do pause loop ;

: green-on   %1 9 lshift GPIOC_BSRR bis!  ;         \
Activated by User pushbutton
: green-off  %1 9 lshift GPIOC_BRR bis!  ;

: blue-on   %1 8 lshift GPIOC_BSRR bis!  ;          \
Activated by Blinky
: blue-off  %1 8 lshift GPIOC_BRR bis!  ;

: meter-high    %1  1 lshift GPIOF_BSRR bis! ;      \ PF-1
high
: meter-low     %1  17 lshift GPIOF_BSRR bis! ;     \ PF-1
low

: blink blue-on .5s-delay blue-off .5s-delay ;

: pb? ( -- ? ) 1 0 lshift GPIOA_IDR bit@            \  test
User pushbutton, lights green led if pressed
if green-on else green-off then pause
;


: loadmeter
    meter-high
    nop
    nop                                             \ some
small delays to extend the meter pulse as this mcu is so fast
    meter-low
    pause
;

task: loadmeter-task
:   load& ( -- )
```

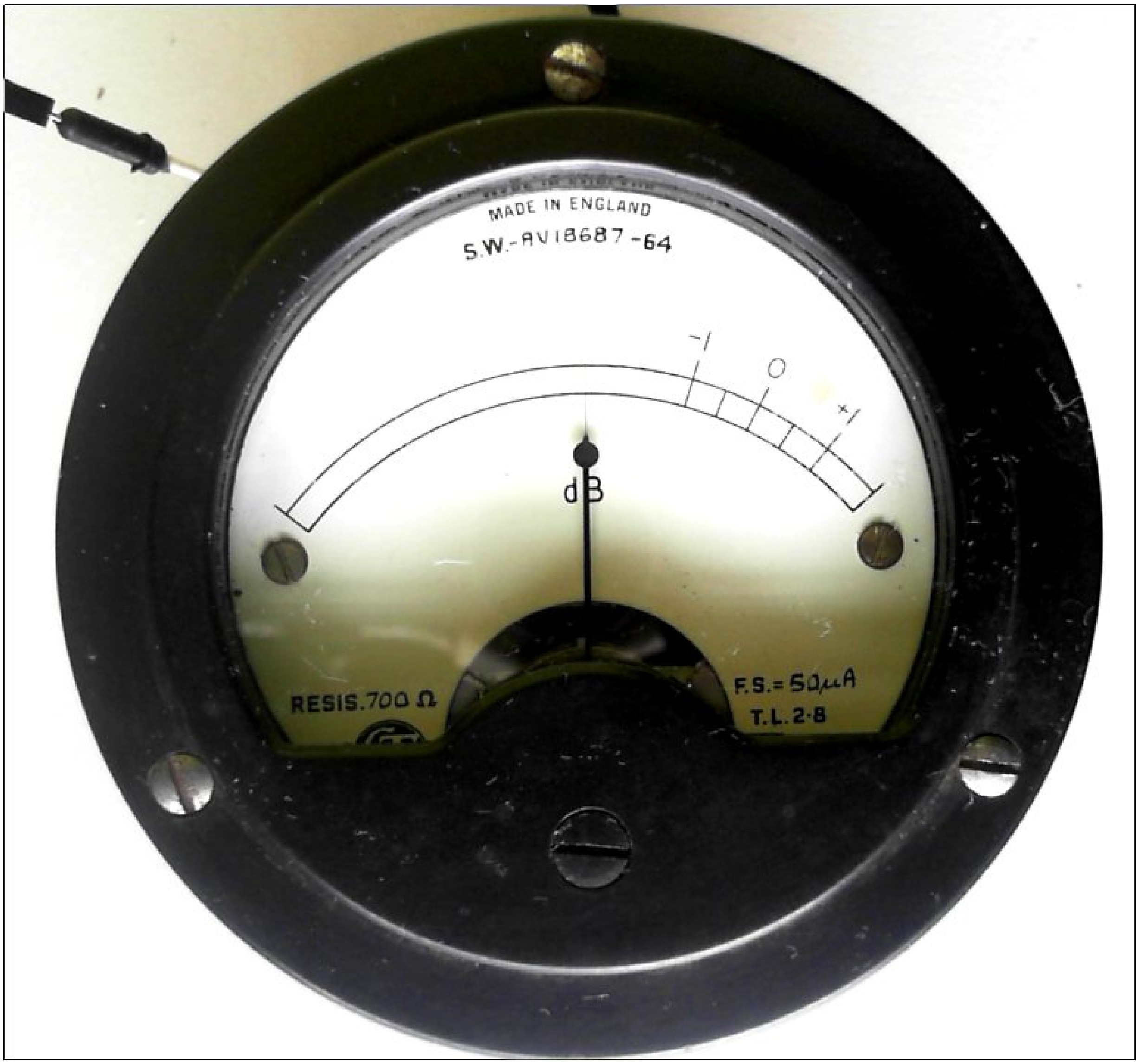
```
    loadmeter-task activate
    begin loadmeter again
;

task: blinktask
:   blinky& ( -- )
    blinktask activate
    begin blink again
;

task: delay1
: delay1& ( -- )
    delay1  activate
    begin delay-1  again
;

task: delay2
: delay2& ( -- )
    delay2  activate
    begin delay-2  again
;

task: user-pb-task
:   user-pb& ( -- )
    user-pb-task activate
    begin  pb? again
;


\ multitask

    load&
    blinky&
    delay1&
    delay2&
    user-pb&
    id                              \ Optional development
word
    mem                             \ Optional development
word
    multitask
```

Loadmeter Pics

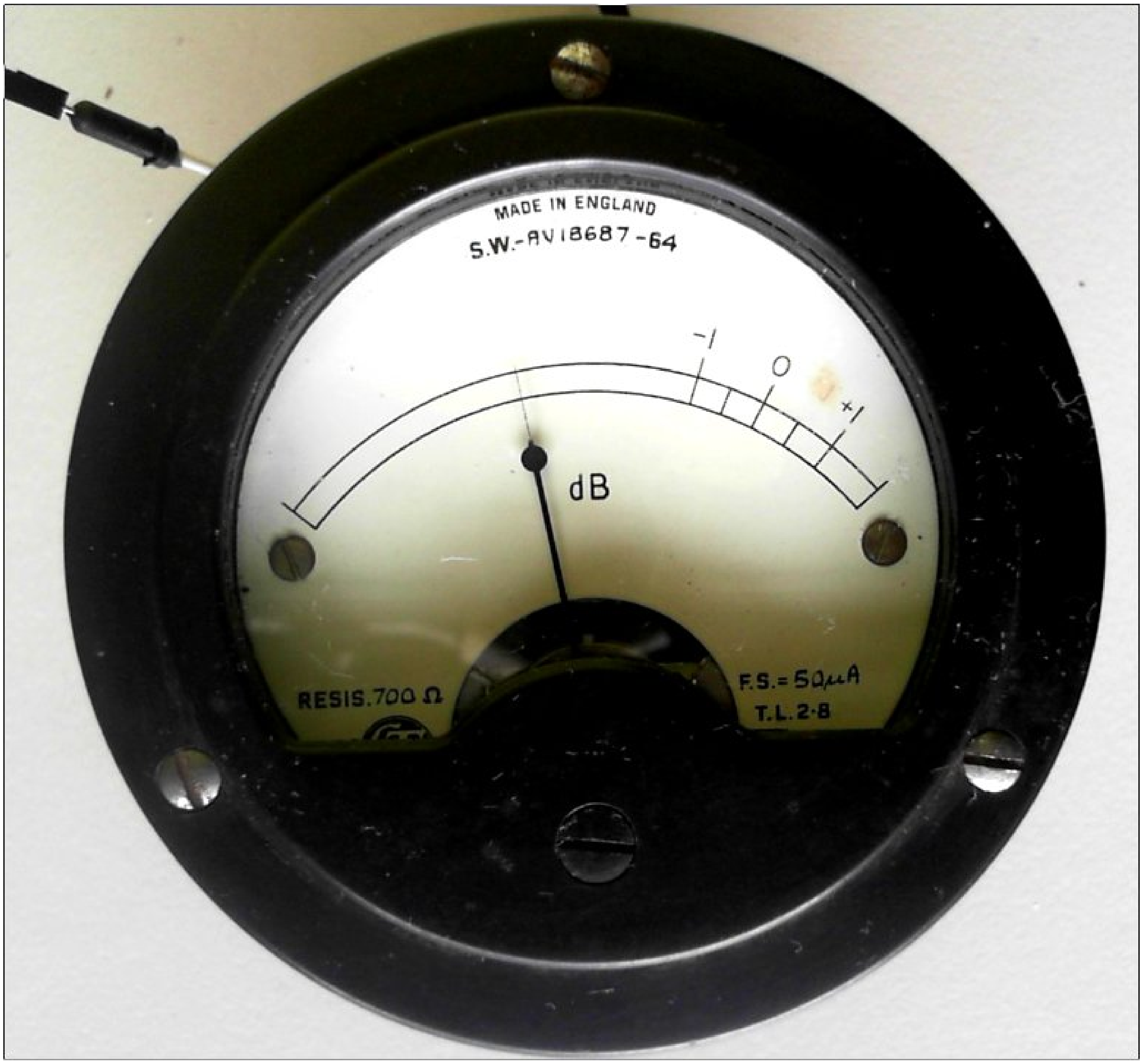
Load task running


MADE IN ENGLAND
S.W.-AV18687-64
-1
0
+1
dB
RESIS.700 Ω
F.S.= 50μA
T.L.2-8

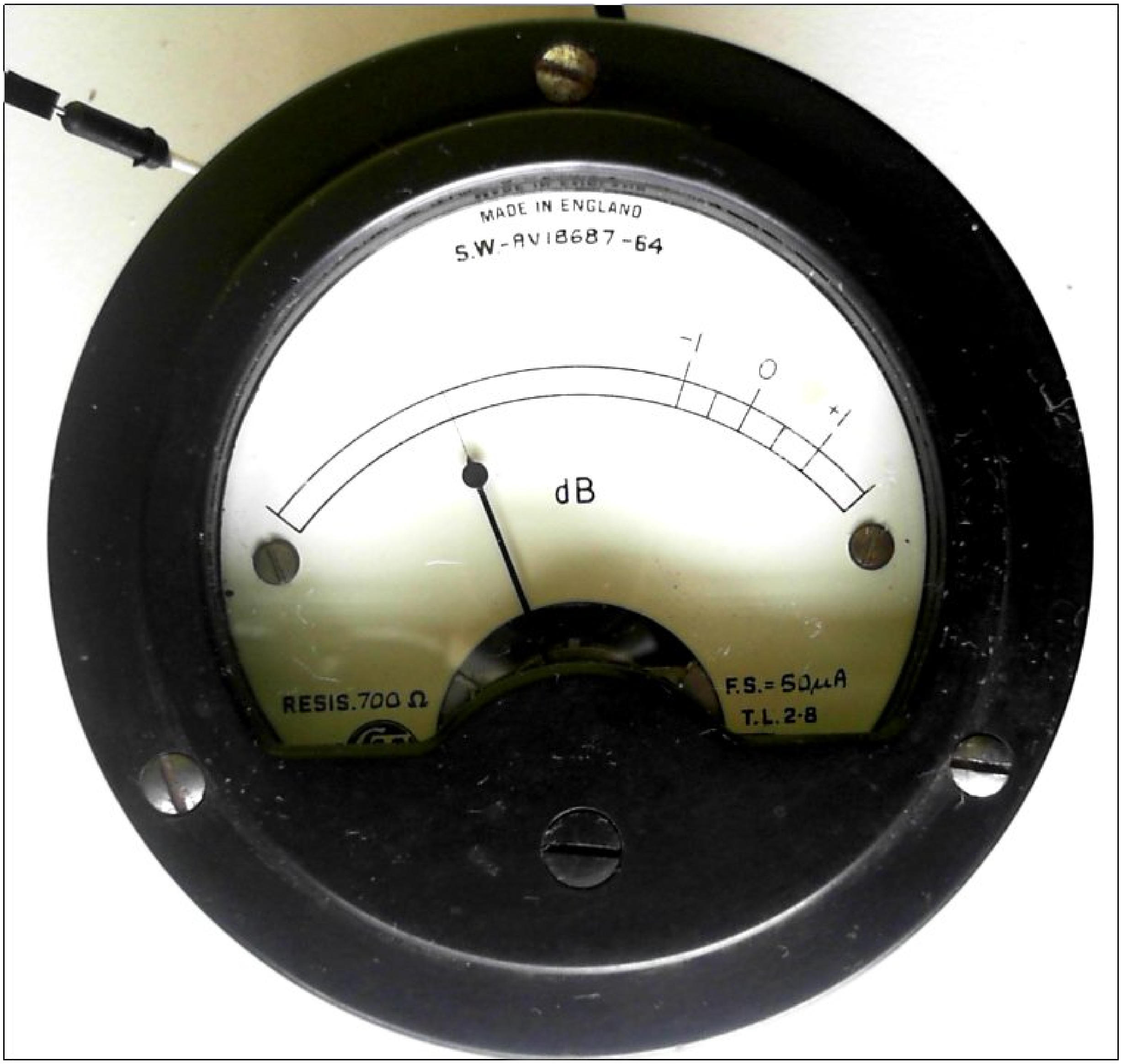
Load and blinky tasks running


MADE IN ENGLAND
S.W.-AV18687-64
-1
0
+1
dB
RESIS.700 Ω
F.S.= 50μA
T.L.2·8

Load, blinky and delay1 tasks running


MADE IN ENGLAND
S.W.-AV18687-64
-1
0
+1
dB
RESIS.700 Ω
F.S.=50μA
T.L.2·8

Load, blinky, delay1, and delay2 tasks running


MADE IN ENGLAND
S.W.-AV18687-64
-1
0
+1
dB
RESIS. 700 Ω
F.S. = 50μA
T.L. 2·8

Load, blinky, delay1, delay2 and user-pb tasks running

MADE IN ENGLAND
S.W.-AV18687-64
-1
0
+1
dB
RESIS.700 Ω
F.S.=50μA
T.L.2·8

Interactive Assembly Delay

This page shows how to use the Mecrisp-Interactive Assembly facility to create a millisecond blocking delay.

Note

While the delay is being used, the CPU is not available to do anything else.

To see the same thing but done by inlining the machine code from arm-none-eabi-as, see Accurate Blocking Delay

Why do it this way ?

The reason that the millisecond blocking delay is done this way rather than in pure Forth is because it can be done in about half the binary code size compared to a pure Forth Word such as “delay” below. Another reason is that sometimes maximum speed is required and this means using minimal instructions in the Forth Word. The delay example is a great way to demonstrate how to achieve this.

Note

The THUMB Instruction Set is used here

Using Pure Forth Instead

The pure Forth way is certainly much faster to write and debug compared to the alternate methods.

The “delay” Word below uses the common “do-loop” twice and is calibrated.

```
: delay
  0 do
     1526 0 do loop
  loop
;
```

Disassembly

```
see delay

20000604: B500  push { lr }
20000606: B430  push { r4  r5 }
20000608: 2400  movs r4 #0
2000060A: 0035  lsls r5 r6 #0
2000060C: CF40  ldmia r7 { r6 }
2000060E: 3F04  subs r7 #4
20000610: 603E  str r6 [ r7 #0 ]
20000612: B430  push { r4  r5 }
20000614: 2400  movs r4 #0
20000616: 25BE  movs r5 #BE
20000618: 00ED  lsls r5 r5 #3
2000061A: 3506  adds r5 #6
2000061C: CF40  ldmia r7 { r6 }
2000061E: 3401  adds r4 #1
20000620: 42AC  cmp r4 r5
20000622: D1FC  bne 2000061E
20000624: BC30  pop { r4  r5 }
20000626: 3401  adds r4 #1
20000628: 42AC  cmp r4 r5
2000062A: D1F0  bne 2000060E
2000062C: BC30  pop { r4  r5 }
2000062E: BD00  pop { pc }

Bytes: 44  ok.
```

Usage

```
1000 delay
```

Produces a 1.0000 second delay.

Interactive Assembly

You need to load the following programs first:

Cortex-M0

1. mecrisp-stellaris-x.x.x/common/disassembler-m0.txt
2. mecrisp-stellaris-x.x.x/common/assembler-m0.txt

Cortex-M3

1. mecrisp-stellaris-x.x.x/common/disassembler-m3.txt
2. mecrisp-stellaris-x.x.x/common/assembler-m0.txt

Table-1

```
l-:                 l+:
jumps               beq
bne                 bcs                 bcc
bmi
bpl                 bvs                 bvc
bhi
bls                 bge                 blt
bgt
ble                 bhs                 blo                 b
wfi                 bl                  ldr=
add                 mov
bx                  blx                 ands
eors                adcs                sbcs
rors
tst                 orrs                muls
bics
mvns                sxtb                sxth
uxtb
uxth                rev                 rev16
revsh
```

```
movs                cmp
adds                subs                str
strb                strh                ldr
ldrb
ldrh                ldrsb               ldrsh
lsls                lsrs                asrs
lslsr               lsrsr               asrsr
push                pop                 stmia
ldmia
```

How does it work ?

Assembler-m0.txt provides the Words in Table-1 which look and behave much like assembler syntax using arm-none-eabi-as. However these are Forth Words and so run under Forth with all the interactive advantages.Forth compiles these Words into the same machine code produced by arm-none-eabi-as.

These Words can be used to produce minimal machine code for inlining in muct the same was as using

Assembly Examples

Using arm-none-eabi-as

Example Blocking Delay Sourcecode

Warning

This code will run perfectly in GDB but it is NO GOOD for use as INLINED MACHINE CODE for Forth because the LDR command produces code that is PC (program counter) relative. To inline machine code in Forth, the code must be PC INDEPENDENT.

```
.syntax unified
.cpu cortex-m0
.thumb
.text
Vector_Table: .word 0x20000000      @ stack pointer value
when stack is empty
ResetVector: .word loop1 +1         @ Reset Handler

loop1:   ldr r0,=1913
loop2:   subs r0,r0,#1
         bne loop2
         subs r6,r6,#1
         bne loop1
```

Disassembly

```
00000008 <loop1>:
   8:  4802            ldr     r0, [pc, #8]    ; (14
<loop2+0xa>)

0000000a <loop2>:
   a:  3801            subs    r0, #1
   c:  d1fd            bne.n   a <loop2>
   e:  3e01            subs    r6, #1
  10:  d1fa            bne.n   8 <loop1>
  12:  07790000        ldrbeq  r0, [r9, -r0]!
```

Using Interactive Assembly

Labels are supported but the syntax and use is different to arm-none-eabi-as.

Example Interactive Assembly Sourcecode

```
: ms-interactive-assy ( u -- ) \ Blocking delay for use on
STM32F0xx @ 8MHz RC Clock. "1000 ms" = 1 second delay
l-:   ldr= r0 1913
l-:   subs r0 #1
      bne -
      subs r6 #1
      bne --
      drop
;
```

Jump to Labels Syntax

Jumping UP:

```
-       Jump to first occurence of  "l-:"
--      Jump to second occurence of "l-:"
---     Jump to third occurence of  "l-:"
```

Jumping Down:

```
+       Jump to first occurence of  "l+"
++      Jump to second occurence of "l+:"
+++     Jump to third occurence of  "l+:"
```

Disassembly

The PUSH and POP are added by Forth

```
see ms-interactive-assy

20000596: B500  push { lr }
20000598: 20EF  movs r0 #EF
2000059A: 00C0  lsls r0 r0 #3
2000059C: 3001  adds r0 #1
2000059E: 3801  subs r0 #1
200005A0: D1FD  bne 2000059E
200005A2: 3E01  subs r6 #1
200005A4: D1F8  bne 20000598
200005A6: CF40  ldmia r7 { r6 }
200005A8: BD00  pop { pc }
```

20000598 to 2000059E is the Forth statement “ldr= r0 1913” computing the shortest MOVS,LSLS,ADDS sequence to guarantee that R0 contains 1913 and is PC INDEPENDENT.

```
movs r0 #EF
lsls r0 r0 #3
adds r0 #1
```

Testing the result:

```
: calc $ef 3 lshift 1 + . cr ;
calc 1913
 ok.
```

Construct the Forth inlined Word from the disassembly above

```
: ms ( u -- )    \ millisecond blocking delay for Cortex-m0
with 8MHz rc clock (mecrisp-stellaris default)
  [              \ 20 bytes size
  $20EF h,
  $00C0 h,
  $3001 h,
  $3801 h,
  $D1FD h,
  $3E01 h,
  $D1F8 h,
  ] drop
;
```

Disassembly

```
see ms

200005E2: B500  push { lr }
200005E4: 20EF  movs r0 #EF
200005E6: 00C0  lsls r0 r0 #3
200005E8: 3001  adds r0 #1
200005EA: 3801  subs r0 #1
200005EC: D1FD  bne 200005EA
200005EE: 3E01  subs r6 #1
200005F0: D1F8  bne 200005E4
200005F2: CF40  ldmia r7 { r6 }
200005F4: BD00  pop { pc }

Bytes: 20  ok.
```

Usage

```
1000 ms
```

Will provide a blocking delay of 1 second.

Interpreter: Inner

Note

Under Construction

Interpreter: Outer

This functionality is provided by the “QUIT” word.

Quit

Reproduced from mecrisp-stellaris-2.5.2/mecrisp-stellaris-source/common/interpreter.s

```
@-----------------------------------------------------------------------------
  Wortbirne Flag_visible| Flag_variable, "hook-quit" @ ( -- addr )
  CoreVariable hook_quit
@-----------------------------------------------------------------------------
  pushdatos
  ldr tos, =hook_quit
  bx lr
  .word quit_innenschleife  @ Simple loop for default

@ ----------------------------------------------------------------------------
  Wortbirne Flag_visible, "quit" @ ( -- )
quit:
@ ----------------------------------------------------------------------------
  @ Endlosschleife - muss LR nicht sichern.  No need for saving LR as this is an endless loop.
  @ Stacks zurücksetzen
  @ Clear stacks and tidy up.
  .ifdef m0core
  ldr r0, =returnstackanfang
  mov sp, r0
  .else
  ldr sp, =returnstackanfang
  .endif

  ldr psp, =datenstackanfang

  .ifdef initflash
   bl initflash
  .endif

  @ Base und State setzen
  ldr r0, =base
  movs r1, #10   @ Base decimal
  str r1, [r0]

  ldr r0, =state
  movs r1, #0    @ Execute mode
  str r1, [r0]

  ldr r0, =konstantenfaltungszeiger
  @ movs r1, #0  @ Clear constant folding pointer
  str r1, [r0]

  ldr r0, =Pufferstand
  @ movs r1, #0  @ Set >IN to 0
  str r1, [r0]

  ldr r0, =current_source
  @ movs r1, #0  @ Empty TIB is source
  str r1, [r0]
  ldr r1, =Eingabepuffer
  str r1, [r0, #4]

quit_intern:
  ldr r0, =hook_quit
  ldr r0, [r0]
  mov pc, r0

quit_innenschleife:  @ Main loop of Forth system.
  bl query
  bl interpret

  .ifdef color

  @ Check state
  ldr r0, =state
  ldr r0, [r0]
  cmp r0, #0
  beq 1f
    write " \x1B[34m"
    b 2f
1:  write " \x1B[36m"
2:

  @ Check memory target
  ldr r0, =Dictionarypointer
  ldr r0, [r0]

  ldr r1, =Backlinkgrenze
  cmp r0, r1
  bhs.n 1f

    writeln "ok'\x1B[0m"
    b.n quit_innenschleife
1:  writeln "ok.\x1B[0m"
    b.n quit_innenschleife

  .else

    writeln " ok."
    b.n quit_innenschleife

  .endif
```

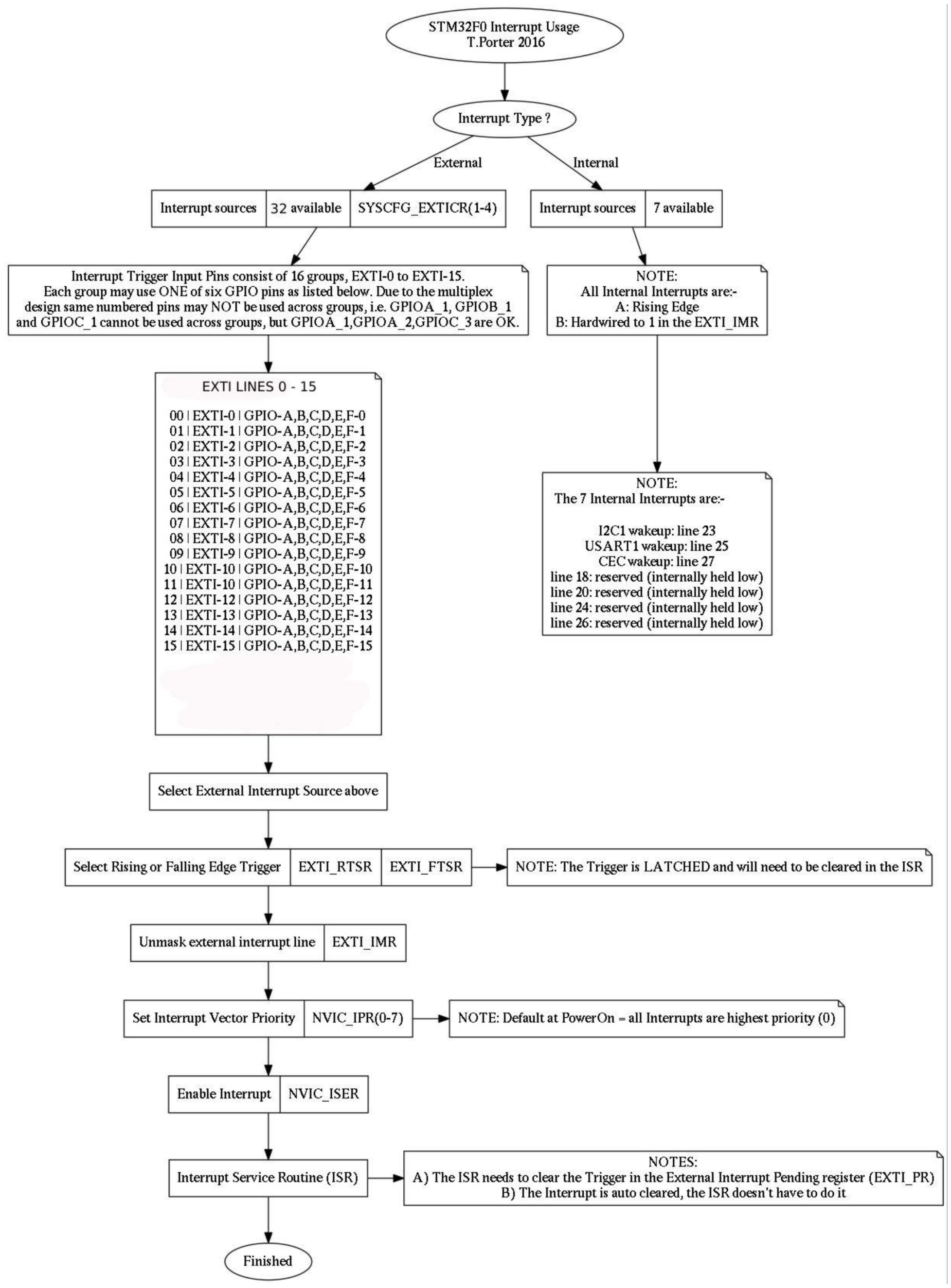
Interrupts

Interrupts, when they occur execute code found in a memory location (which is reserved for that interrupt), save the state of the MCU, execute the code, then restore the MCU state. There are many other optional factors affecting interrupts, such as priority, triggers and so on.

Note

- In the Cortex-M programming manual, an Exception is anything that breaks the normal program flow, and invokes a handler from the vector table. Interrupts are a subset of Exceptions, coming from the peripherals outside the ARM core.
- Exceptions have an Exception Number, starting from 0.
- Interrupts have an IRQ Number, starting from 0.
- Because all Interrupts are Exceptions, they all get an Exception Number, which is 16 higher than the IRQ Number. The Exceptions that are not Interrupts (including SysTick) have Exception Numbers in the 0-15 range, smaller than the Exception Numbers of the Interrupts. Somewhat confusingly, Exceptions that are not Interrupts have IRQ Numbers too, which by extension fall into the range from -16 to -1.
- Because SysTick is implemented in the Cortex-M core, it is considered an exception, but not an interrupt having a Exception Number of 15, and IRQ Number -1.

Cortex M0 Interrupt Flowchart


STM32F0 Interrupt Usage
T.Porter 2016
Interrupt Type ?
External
Internal
Interrupt sources
32 available
SYSCFG_EXTICR(1-4)
Interrupt sources
7 available
Interrupt Trigger Input Pins consist of 16 groups, EXTI-0 to EXTI-15.
Each group may use ONE of six GPIO pins as listed below. Due to the multiplex
design same numbered pins may NOT be used across groups, i.e. GPIOA_1, GPIOB_1
and GPIOC_1 cannot be used across groups, but GPIOA_1,GPIOA_2,GPIOC_3 are OK.
NOTE:
All Internal Interrupts are:-
A: Rising Edge
B: Hardwired to 1 in the EXTI_IMR
EXTI LINES 0 - 15
00 | EXTI-0 | GPIO-A,B,C,D,E,F-0
01 | EXTI-1 | GPIO-A,B,C,D,E,F-1
02 | EXTI-2 | GPIO-A,B,C,D,E,F-2
03 | EXTI-3 | GPIO-A,B,C,D,E,F-3
04 | EXTI-4 | GPIO-A,B,C,D,E,F-4
05 | EXTI-5 | GPIO-A,B,C,D,E,F-5
06 | EXTI-6 | GPIO-A,B,C,D,E,F-6
07 | EXTI-7 | GPIO-A,B,C,D,E,F-7
08 | EXTI-8 | GPIO-A,B,C,D,E,F-8
09 | EXTI-9 | GPIO-A,B,C,D,E,F-9
10 | EXTI-10 | GPIO-A,B,C,D,E,F-10
11 | EXTI-10 | GPIO-A,B,C,D,E,F-11
12 | EXTI-12 | GPIO-A,B,C,D,E,F-12
13 | EXTI-13 | GPIO-A,B,C,D,E,F-13
14 | EXTI-14 | GPIO-A,B,C,D,E,F-14
15 | EXTI-15 | GPIO-A,B,C,D,E,F-15
NOTE:
The 7 Internal Interrupts are:-
I2C1 wakeup: line 23
USART1 wakeup: line 25
CEC wakeup: line 27
line 18: reserved (internally held low)
line 20: reserved (internally held low)
line 24: reserved (internally held low)
line 26: reserved (internally held low)
Select External Interrupt Source above
Select Rising or Falling Edge Trigger
EXTI_RTSR
EXTI_FTSR
NOTE: The Trigger is LATCHED and will need to be cleared in the ISR
Unmask external interrupt line
EXTI_IMR
Set Interrupt Vector Priority
NVIC_IPR(0-7)
NOTE: Default at PowerOn = all Interrupts are highest priority (0)
Enable Interrupt
NVIC_ISER
Interrupt Service Routine (ISR)
NOTES:
A) The ISR needs to clear the Trigger in the External Interrupt Pending register (EXTI_PR)
B) The Interrupt is auto cleared, the ISR doesn't have to do it
Finished

NVIC

NESTED VECTORED INTERRUPT CONTROLLER

NVIC main features

- 32 maskable interrupt channels (not including the sixteen Cortex® -M0 interrupt lines)
- 4 programmable priority levels (2 bits of interrupt priority are used)

- Low-latency exception and interrupt handling
- Power management control
- Implementation of System Control Registers

The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts.

All interrupts including the core exceptions are managed by the NVIC. For more information on exceptions and NVIC programming, refer to the PM0215 programming manual.

TABLE B: Interrupt and Exception Vectors

Each Interrupt Source matches a “Position” value (see explanation below) and these are used to ENABLE the ISR. How it works is shown in the example programs segments.

Note

Depending on MCU your device may not have all the peripherals listed in this table, or may have many more.

Position	Priority	Type of Priority	Interrupt Source	Description	Address
.	.	.	.	.	0x0000 0000
.	-3	fixed	Reset	Reset	0x0000 0004
.	-2	fixed	NMI	Non maskable interrupt.	0x0000 0008
.	-1	fixed	HardFault	All class of fault	0x0000 000C

Position	Priority	Type of Priority	Interrupt Source	Description	Address
.	3	settable	SVCall	System service call via SWI instruction	0x0000 002C
.	5	settable	PendSV	Pendable request for system service	0x0000 0038
.	6	settable	SysTick	System tick timer (SPECIAL SEE ABOVE)	0x0000 003C
.	7	settable	WWDG	Window watchdog interrupt	0x0000 0040
1	8	settable	PVD_VDDIO2	PVD and VDDIO2 supply comparator interrupt (combined EXTI lines 16 and 31)	0x0000 0044
2	9	settable	RTC	RTC interrupts (combined EXTI lines 17, 19 and 20)	0x0000 0048

Position	Priority	Type of Priority	Interrupt Source	Description	Address
3	10	settable	FLASH	Flash global interrupt	0x0000 004C
4	11	settable	RCC_CRS	RCC and CRS global interrupts	0x0000 0050
5	12	settable	EXTI0_1	EXTI Line[1:0] interrupts	0x0000 0054
6	13	settable	EXTI2_3	EXTI Line[3:2] interrupts	0x0000 0058
7	14	settable	EXTI4_15	EXTI Line[15:4] interrupts	0x0000 005C
8	15	settable	TSC	Touch sensing interrupt	0x0000 0060
9	16	settable	DMA_CH1	DMA channel 1 interrupt	0x0000 0064
10	17	settable	DMA_CH2_3	DMA channel 2 and 3 interrupts	0x0000 0068
10	17	settable	DMA2_CH1_2	DMA2 channel 1 and 2 interrupts	0x0000 0068

Position	Priority	Type of Priority	Interrupt Source	Description	Address
11	18	settable	DMA_CH4_5_6_7	DMA channel 4, 5, 6 and 7 interrupts	0x0000 006C
11	18	settable	DMA2_CH3_4_5	DMA2 channel 3, 4 and 5 interrupts	0x0000 006C
12	19	settable	ADC_COMP	ADC and COMP interrupts (ADC interrupt combined with EXTI lines 21 and 22)	0x0000 0070
13	20	settable	TIM1_BRK_UP_TRG_COM	TIM1 break, update, trigger and commutation interrupt	0x0000 0074
14	21	settable	TIM1_CC	TIM1 capture compare interrupt	0x0000 0078
15	22	settable	TIM2	TIM2 global interrupt	0x0000 007C
16	23	settable	TIM3	TIM3 global interrupt	0x0000 0080

Position	Priority	Type of Priority	Interrupt Source	Description	Address
17	24	settable	TIM6_DAC	TIM6 global interrupt and DAC underrun interrupt	0x0000 0084
18	25	settable	TIM7	TIM7 global interrupt	0x0000 0088
19	26	settable	TIM14	TIM14 global interrupt	0x0000 008C
20	27	settable	TIM15	TIM15 global interrupt	0x0000 0090
21	28	settable	TIM16	TIM16 global interrupt	0x0000 0094
22	29	settable	TIM17	TIM17 global interrupt	0x0000 0098
23	30	settable	I2C1	I2C1 global interrupt (combined with EXTI line 23)	0x0000 009C
24	31	settable	I2C2	I2C2 global interrupt	0x0000 00A0
25	32	settable	SPI1	SPI1 global interrupt	0x0000 00A4

Position	Priority	Type of Priority	Interrupt Source	Description	Address
26	33	settable	SPI2	SPI2 global interrupt	0x0000 00A8
27	34	settable	USART1	USART1 global interrupt (combined with EXTI line 25)	0x0000 00AC
28	35	settable	USART2	USART2 global interrupt (combined with EXTI line 26)	0x0000 00B0
29	36	settable	USART3_4	USART3 and USART4 global interrupts	0x0000 00B4
30	37	settable	CEC_CAN	CEC and CAN global interrupts (combined with EXTI line 27)	0x0000 00B8
31	38	settable	USB	USB global interrupt (combined with EXTI line 18)	0x0000 00BC

Position ?

“Position” is the number supplied to the “IMPORTANT NVIC examples” below.

In this Cortex-M0 table it stops at 31, but other Cortex-M MCUs can have “Position” values numbering into the hundreds.

Nvic_iser

The NVIC_ISERx registers are dual-purpose. If you write a 1 to a specific bit it will enable the interrupt. If you read a specific bit it will tell you if the interrupt is enabled.

If a pending interrupt is enabled the NVIC activates the interrupt based on its priority. If an interrupt is not enabled asserting its interrupt signal changes the interrupt state to pending but the NVIC never activates the interrupt regardless of its priority.

If the “pending” flag gets set then the interrupt will be served as soon as you re-enable the interrupt.

Warning

NVIC numbers 0 - 31 are accessed by NVIC_ISER0_SETENA. NVIC numbers 32 - 63 are accessed by NVIC_ISER1_SETENA, and so on. This is mentioned in only ONE place in the reference,

IMPORTANT NVIC examples

In the first example, the Binary number %1000000 equates to “Position” 6 as Bit 6 is HIGH and NVIC_ISER0_SETENA is used. NVIC_ISER0_SETENA offers “Positions” from 0 - 31.

In the second example, the Binary number %100000000 equates to “Position” 40 as Bit 8 is HIGH and NVIC_ISER1_SETENA is used. “Position” 40 isn’t available on a Cortex-M, but this example is taken from a STM32F103 Cortex-M3 MCU which has NVIC “Positions” up to number 67. NVIC_ISER1_SETENA offers “Positions” from 32 - 63.

```
%1000000  NVIC_ISER0_SETENA     \ enable interrupt #6
%100000000 NVIC_ISER1_SETENA    \ enable interrupt #40.  40 = 32 + 8
```

EXTI

EXTERNAL INTERRUPT/EVENT CONTROLLER

The external interrupt/event controller consists of up to 20 edge detectors in connectivity line devices, or 19 edge detectors in other devices for generating event/interrupt requests. Each input line can be independently configured to select the type (event or interrupt) and the corresponding trigger event (rising or falling or both). Each line can also masked independently. A pending register maintains the status line of the interrupt requests

EXTI controller main features

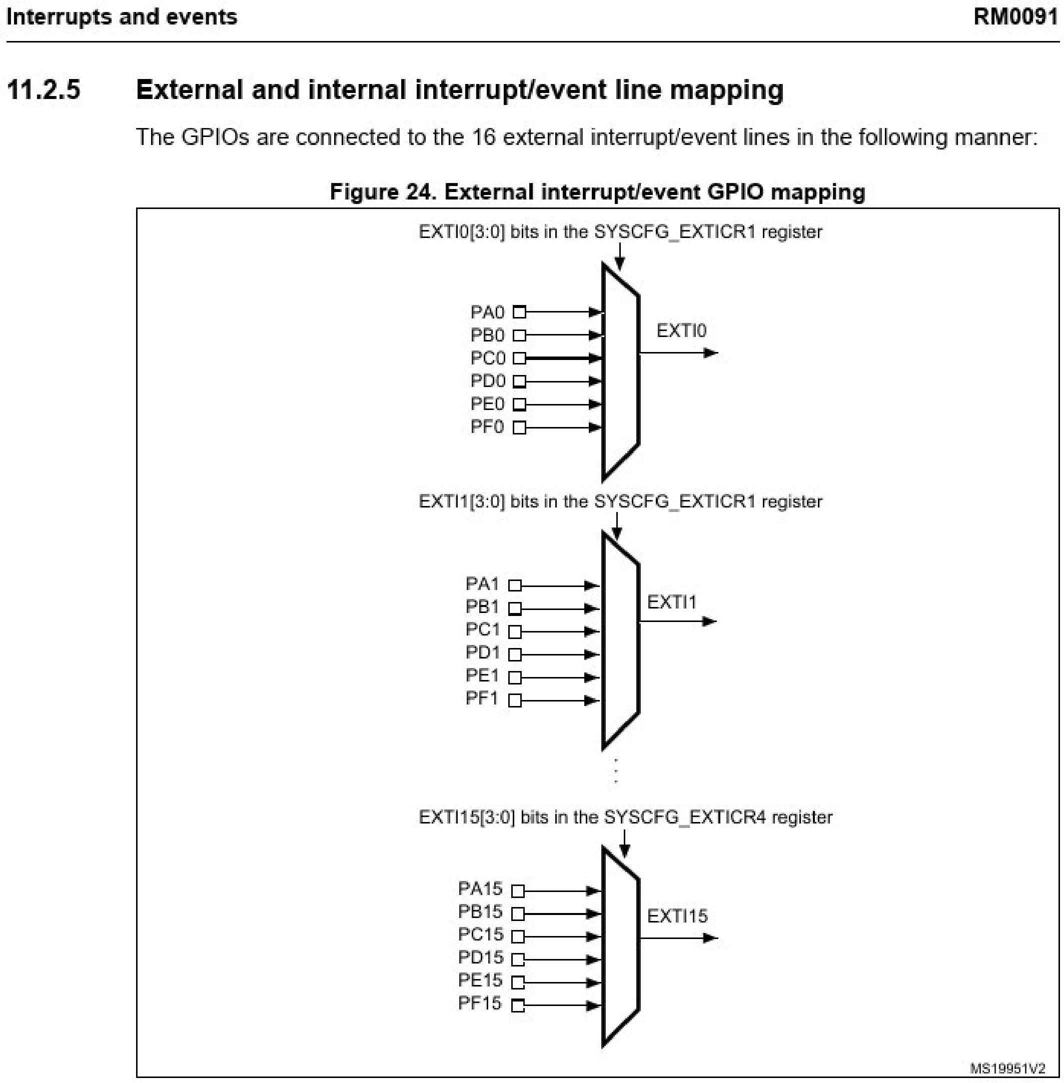
- Independent trigger and mask on each interrupt/event line
- Dedicated status bit for each interrupt line
- Generation of up to 20 software event/interrupt requests
- Detection of external signal with pulse width lower than APB2 clock period. Refer to the electrical characteristics section of the datasheet for details on this parameter.

EXTI Lines 0 - 15

Used by the GPIO pins

TABLE A: GPIO EXTI Mapping

There are 16 ‘EXTI lines’ (EXTI0-15). Each line is driven by the same numbered bit of all GPIOs. As can be seen, six GPIOS drive the same bit numbered ‘EXTI line’ but only one GPIO can be selected per EXTI.



Interrupts and events RM0091

11.2.5 External and internal interrupt/event line mapping

The GPIOs are connected to the 16 external interrupt/event lines in the following manner:

Figure 24. External interrupt/event GPIO mapping

EXTI0[3:0] bits in the SYSCFG_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PF0
EXTI0
EXTI1[3:0] bits in the SYSCFG_EXTICR1 register
PA1
PB1
PC1
PD1
PE1
PF1
EXTI1
EXTI15[3:0] bits in the SYSCFG_EXTICR4 register
PA15
PB15
PC15
PD15
PE15
PF15
EXTI15
MS19951V2

EXTI Lines 16 - 31

Used by the Peripherals

EXTI Line	Connected To
EXTI line 16	PVD output

EXTI Line	Connected To
EXTI line 17	RTC Alarm event
EXTI line 18	internal USB wakeup event
EXTI line 19	RTC Tamper and TimeStamp events
EXTI line 20	RTC Wakeup event (available only on STM32F07x and STM32F09x devices)
EXTI line 21	Comparator 1 output
EXTI line 22	Comparator 2 output
EXTI line 23	internal I2C1 wakeup event
EXTI line 24	reserved (internally held low)
EXTI line 25	internal USART1 wakeup event
EXTI line 26	internal USART2 wakeup event (available only on STM32F07x and STM32F09x devices)
EXTI line 27	internal CEC wakeup event

EXTI Line	Connected To
EXTI line 28	internal USART3 wakeup event (available only on STM32F09x devices)
EXTI line 29	reserved (internally held low)
EXTI line 30	reserved (internally held low)
EXTI line 31	VDDIO2 supply comparator output (available only on STM32F04x, STM32F07x and STM32F09x devices

Hardware Interrupt Selection

To configure a line as interrupt source, use the following procedure:

1. Choose one of the 16 GPIO EXTI mapping groups to suit your GPIO pin from TABLE A above.
2. Choose the GPIO for the selected EXTI.

SEL	GPIO
x000:	PA[x] pin
x001:	PB[x] pin
x010:	PC[x] pin
x011:	PD[x] pin
x100:	PE[x] pin

SEL	GPIO
x101:	PF[x] pin

I.E. GPIOC-2 would be:

```
%X010  8  lshift $40010008  bis!  \ SYSCFG_EXTICR1_EXTI2
```

Warning

To configure any SYSCFG registers like the one above, SYSCFGEN must be enabled in the RCC Register. If it isn’t then **only GPIOA works** as a trigger for EXTI !

3. Configure the interrupt mask bit in the EXTI_IMR register.
4. Configure the Trigger Selection bits of the Interrupt line (EXTI_RTSR and EXTI_FTSR)
5. Configure the enable and mask bits that control the NVIC IRQ channel mapped to the EXTI used in 1) above.
6. Set the priority for the interrupt vector in question in the NVIC through the IPR0-IPR7 registers.
7. Enable the interrupt in the NVIC_ISER_SETENA, use the Bit Position from TABLE B to suit your Interrupt Source. If it’s a EXTI Line Interrupt group, then it will be Bit Position 5 ($20), 6 ($40) or 7 ($80).
8. Write your interrupt service routine (ISR)

Hardware event selection

To configure a line as event source, use the following procedure:

```
Configure the corresponding mask bit in the EXTI_EMR register.
Configure the Trigger Selection bits of the Event line (EXTI_RTSR and EXTI_FTSR)
```

Software interrupt/event selection

Any of the external lines can be configured as software interrupt/event lines. The following is the procedure to generate a software interrupt.

```
Configure the corresponding mask bit (EXTI_IMR, EXTI_EMR)
Set the required bit of the software interrupt register (EXTI_SWIER)
```

Interrupt Service Routine

Inside your interrupt service routine, check the source of the interrupt…either the GPIO pin directly or the external interrupt line. Once you figure out which one triggered the interrupt, perform the interrupt processing scheme associated with it. Make sure that you clear the corresponding pending bit of the external interrupt lines of interest in the EXT_PR (external interrupt pending register) register by writing a ‘1’ to it. [hertaville]

Forth Interrupt Methodology

Mecrisp-Stellaris uses the “tick” (‘) Word to obtain the memory location of the “Interrupt Handler” Word or ISR to be executed when the interrupt occurs via a ‘interrupt hook’.

```
['] interrupt-handler irq-exti0_1 !        \ See the example programs for more
information
```

“interrupt-handler” is the ISR, but where does the “irq-exti0_1” interrupt hook come from ?

It’s in the Dictionary for your chip, listed with all the available interrupt hooks. The interrupt hooks for a STM32F051 are listed here.

```
irq-systick
irq-fault           irq-collection      irq-exti0_1         irq-exti2_3
irq-exti4_15        irq-tsc             irq-dma_ch1         irq-dma_ch2_3
irq-dma_ch4_5       irq-adc             irq-tim1_up         irq-tim1_cc
irq-tim2            irq-tim3            irq-tim6_dac        irq-tim14
irq-tim15           irq-tim16           irq-tim17           irq-i2c1
irq-i2c2            irq-spi1            irq-spi2            irq-usart1
irq-usart2          irq-cec_can
```

Interrupt Examples

```
: pb.interrupt.init ( #6 )
  0 AFIO_EXTICR1_EXTI0                   \ line PA0
  line.0.unmask                          \ unmask line 0
  EXTI_RTSR_TR0                          \ rising trigger in line 0
  ['] pb.interrupt.handler irq-exti0 !   \ link exti0 to pb.interrupt.handler
  %1000000  NVIC_ISER0_SETENA            \ enable interrupt #6
;

: lmt01.interrupt.init ( #40 )
  1 AFIO_EXTICR4_EXTI12                           \ line PB12
  line.12.unmask                                  \ unmask line 12
  EXTI_FTSR_TR12                                  \ falling trigger on line 12
  ['] lmt01.interrupt.handler irq-exti10 !        \ link exti10 ( to
pb.interrupt.handler
  %100000000 NVIC_ISER1_SETENA                    \ enable enterrupt #40
;
```

Systick is special!

Systick, being a Cortex-M core, and not a STM32F peripheral has its own set-pending and clear-pending bits in the SCB_ICSR as below.

```
SCB_ICSR () $00000000
INTERRUPT CONTROL AND STATE REGISTER
Provides:
*A set-pending bit for the Non-Maskable Interrupt NMI exception
*Set-pending and clear-pending bits for the PendSV and SysTick
exceptions.
Indicates:
*The exception number of the exception being processed
*Whether there are preempted active exceptions
*The exception number of the highest priority pending exception
*Whether any interrupts are pending.
N                 I
M     P P P P     S
I     E E E E     R
P     N N N N     P
E     D D D D     E
N     S S S S     N
D     V V T T     D
S     S C S C     I
E     E L E L     N
T     T R T R     G        |------VECT PENDING-----| VECT ACTIVE
3|    | 2| 2| 2| 2|    | 2|         | 1| 1| 1| 1| 1| 1|            | -----------
1| ~~~| 8| 7| 6| 5| ~~~| 2| ~~~~~~~| 7| 6| 5| 4| 3| 2| ~~~~~~~~~~~| 5| 4| 3| 2| 1| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Intro: Forth

When was Forth Created ?

FORTH was created back in 1968 on an IBM 1130 by Charles Moore

> **Note**
>
> Programming in Forth is generating higher levels of abstractions, until you have a language well fitted to solve your problem. Forth strongly encourages the hacker’s paradigm of “plan one to throw away”. It is often cheaper to rewrite a program once you understood what you really wanted instead of living with something that is broken by design. The assumption that it just needs someone clever enough to do the design right in the first place usually is void - noone can look into the future, and without the experience of using the beast, nobody really knows what to put in. - Bernd Paysan

Is Forth Interpreted or Compiled ?

It’s important to be clear: while forth systems are certainly interactive and have an interpreter, definitions of forth words (procedures) are almost always compiled as they are read in. Forth programs run at close to machine code speed. (Usually a factor of ~ 2 slower due to the use of threaded code, but this can be made up for in other ways.) Forth is very close to the metal.

It’s actually more complicated than this (of course!) because forth words (procedures) can themselves control the parsing and compilation process in ways that will make your head spin. This is part of what makes forth incredibly fun and powerful. Read Starting Forth and an implementation like JONESFORTH for more information. - Tim @ hackaday.com

The Forth Philosophy

If Forth has a credo, it might be summed up as smallness, simplicity, and speed. Forth programmers traditionally disdain the flashy user interfaces and elaborate error handling that characterize most integrated environments nowadays. This attitude has its benefits ” for example, a self-contained interactive Forth system including an editor, assembler, and even multitasking support can easily be put in an 8 KB EPROM. It also has its drawbacks ” runtime error checking in a Forth system is virtually absent (unless it happens to come “for free,” such as divide-by-zero detection on the Intel 80x86 family). Frequent total system crashes during Forth program development are taken for granted; indeed, they are expected.

Corollaries to the Forth credo are that intimate access to the hardware should always be available and portability should never get in the way of exploiting a CPU’s instruction set to its fullest. A Forth system without an assembler is felt by most to be only slightly better than no Forth system at all. As for portability, you would be amazed at the number of arguments in the ANSI Forth Technical Committee that have hinged on whether a particular language feature can be implemented efficiently on someone’s favorite processor.

Classic Forth programming style is characterized by many, many short, relatively simple definitions, bottom-up design, prototyping, and successive approximations. Forth dogma has it that an ideal definition should not exceed 2 or 3 lines and should not contain more than 9 or 10 elements. This ensures that individual definitions can be exhaustively tested; the programmer can simply put parameters on the stack and exercise the definition interactively through all of its possible internal paths. It also constrains an individual definition from becoming too specific, so that it is more likely to be reusable in the application.

The bottom up design of a Forth application derives naturally from the types of problems to which the language is routinely applied. A typical Forth application involves intimate interaction with hardware to acquire data or control a machine; in many cases, the hardware itself is only poorly understood or is not yet stable. The Forth words which are most hardware dependent are thus usually coded first; when these definitions are proven and the hardware is well understood, the rest of the application is built up by adding layers of increasing sophistication and complexity.

The emphasis on prototyping and successive approximations has much the same grounds as the stress on bottom-up design. In most Forth applications, getting the hardware to do what you want puts you a long ways toward your goal. After prototyping some code that approximates the final form of the application, you can simply discard or modify the upper layers until you get the specific functionality and user interface that you need; the lower, more hardware dependent levels of the application rarely require any significant change.

An important advantage of Forth in application design is that the documented user interface can rely on Forth’s own interpreter, and merely consist of the top layer of definitions. This offers the sophisticated user incredible flexibility, because not only can he use the documented application entry points, he can extend the application by putting building blocks together in new ways, or access lower level words to test individual hardware components. In environments with naive users, the Forth compiler and the interpretation of “dangerous” Forth words are disabled, or an application-specific, self-contained user interface is written that does not lean on the Forth interpreter.

(source: Originally published as “A Forth Apologia” in Programmer’s Journal, Volume 6, Number 6, November/December 1988, page 56.)

Forth: strengths and weaknesses

Forth is the most obvious language to consider using on a stack machine. That is because the Forth language is based upon a set of primitives that execute on a virtual stack machine architecture. All the stack machines presented in this book support highly efficient implementations of Forth. All of these machines can use a Forth compiler to generate efficient machine code. The biggest advantage of using Forth then, is that the highest processing rate possible can be squeezed from the machine.

One of the characteristics of Forth is its very high use of subroutine calls. This promotes an unprecedented level of modularity, with approximately 10 instructions per procedure being the norm. Tied in with this high degree of modularity is the interactive development environment used by Forth compilers. In this environment programs are designed from the top down, using stubs as appropriate. Then they are built from the bottom up, testing each and every short procedure interactively as it is written. On large projects, the top-down and bottom-up phases are repeated in cycles.

Since Forth is a stack-based, interactive language, no testing programs or “scaffolding” need be written. Instead, the values to be passed to the procedure are pushed onto the stack from the keyboard, the procedure to be tested is executed, and the results are returned on the top of the stack. This interactive development of modular programs is widely claimed by experienced Forth programmers to result in a factor of 10 improvement in programmer productivity, with improved software quality and reduced maintenance costs. Part of this gain may come from the fact that Forth programs are usually quite small compared to equivalent programs in other languages, requiring less code to be written and debugged. A 32K byte Forth program, exclusive of symbol table information, is considered a monster, and may take several hundred thousand bytes of source code to generate.

One of the advantages of the Forth programming language is that it covers the full spectrum of language levels. Some languages, such as assembly language, allow dealing only at the hardware level. Other languages, such as FORTRAN, deal at an abstract level that has little to do with the underlying machine. Forth programs can span the full range of programming abstraction. At the lowest level, Forth allows direct access to hardware ports in the system for real time I/O handling and interrupt servicing. At the highest level, the same Forth program can manage a sophisticated knowledge base.

The one facet of Forth that is most interesting (and baffling to many casual observers) is that it is an extensible language. As every procedure is added to the language, the apparent language available to the programmer grows. In this manner, Forth is much like LISP. There is no distinction made between core procedures in the language and extensions added by the programmer. This enables the language to be flexible to an extent beyond comprehension to people who have not extensively used the capability.

The extensibility of Forth does have mixed blessings. Forth tends to act as a programmer amplifier. Good programmers become exceptional when programming in Forth. Excellent programmers can become phenomenal. Mediocre programmers generate code that works, and bad programmers go back to programming in other languages. Forth also has a moderately difficult learning curve, since it is different enough from other programming languages that bad habits must be unlearned. New ways of conceptualizing solutions to problems must be acquired through practice. Once these new skills are acquired, though, it is a common experience to have Forth-based problem solving skills involving modularization and partitioning of programs actually improve a programmer's effectiveness in other languages as well.

Another problem with some Forth systems is that they do not include a rich enough set of programming tools to suit many programmers. Also, older Forth systems cooperate poorly with resident operating systems. These traits stem from Forth's history of use on very small machines with few hardware resources. In real time control applications, these limitations are generally not much of a problem. Other applications need better support tools. Fortunately, the trend is for newer Forth systems to provide much better development environments and library support than in the past.

The result of all these effects is that Forth is best used on medium-sized programming projects involving no more than two or three programmers who have compatible programming styles. In any very large programming project, clashing styles and abilities tend to prevent the production of extremely high quality software. However, within these constraints, Forth programs are consistently delivered in a very short time with excellent results, often solving problems that could not be solved in any other language, or at least, not solved within budget and development time constraints.

(source: https://users.ece.cmu.edu/~koopman/stack_computers/sec7_2.html#721)

Mecrisp-Stellaris Forth

Mecrisp Stellaris Forth is a port of Mecrisp Forth to the ARM Cortex M architecture written by Matthias Koch. They share most of the design ideas, so Mecrisp-Stellaris can compile directly into Flash, generates native code with constant folding and inlining of short words.

- Mecrisp Stellaris Homepage: https://sourceforge.net/projects/mecrisp/

These docs are maintained by <terry AT tjporter d0t com D0t au>, a happy Mecrisp-Stellaris user, and are not the official docs which are available at the Mecrisp Stellaris Homepage.

If you would like to assist with these docs, read the ‘How to Contribute to the UserDoc_Project’ section, and send me your patches. If you send me enough useful patches, I’ll give you developer rights. All contributors will be listed in the ‘contributors’ section of these docs.

Jump To Far

One very annoying error to encounter is the “jump too far”, what does it mean and how is it avoided ?

Error Description

This error is an artifact of the Cortex-M Thumb Instruction Set Architecture which limits the Program Counter jumping further than 1024 locations from the current Program Counter Value.

Example

When you need one, one can’t be found!

<waiting until I have a example error> @ 25 Dec 2019>

Kernel Deep Configurations

Kernel Config File Name?

mecrisp-stellaris-source/common/forth-core.s

Stack Size

Increase the Data and Return stack sizes.

Note

Defaults are 64 elements each.

Allocation is completely static and preserves locations across Reset.

Levels of Code in Forth Programming

A essay by the late Jeff Fox

Sourced: http://www.ultratechnology.com/levels.htm on 30/07/2019

Levels of code in Forth programming

I had been programming in Fortran and assembler for a number of years when I got my first personal computer. It was exciting. On the first generation PC one had to toggle in a paper tape loader from front panel switches, byte by byte, then start the paper tape reader and then one could output to the led on the front panel. I had waited for the first model that booted from ROM, had a parallel port for keyboard input and had video output from he factory, it was 1974. The PC I got had a wonderful monitor program that replaced a front panel with a nice video display of the contents of all the CPU registers where one could change anything and single-step through programs seeing everything visualized before you. One could also the video for text and graphics display by your programs.

I found it very educational. About a year later I upgraded from 1KB of RAM to 9KB and could now run an assembler or BASIC interpreter/editor on the 8080. There was DOS but I could not afford a disk and used cassette storage. I also found FIG-FORTH about that time. Since I had assembler training I found that Forth let me do everything I could in assembler and much more and do it much faster and re-use my old code that I had compiled and added to the dictionary in my-language.

I started with voice and speaker recognition, voice output, and then I added graphics and sound and 3D visualization with interactive simulations of objects and access to the sub-logic/MS 3D databases extensions to FIG-FORTH and sold my first system to NASA. I found porting the whole environment fairly trivial from a cassette based environment on 8080 to a better graphic and sound environment on 6502 with block based disk I/O access. I had been using Forth quite a bit now and starting to learn more about it.

My Forth code was carefully layered. I had actually modified the ROMs on my first machine to default into loading Forth. I had access from the ROMs on up. I could use ROM code from RAM or write my own CODE (assembler) words in Forth to form the lowest level of software above the hardware. I had a layer of code of this type for the registers, memory, and I/O ports. In some cases there was a layer of code to define a serial port or other abstracted device on some parallel port pins using a timer or something. Then there was a layer of code to provide abstracted control over all the abstracted hardware and software devices. A layer for the Forth kernel, a layer for the high level Forth extensions, a layer for my 3D and sound and voice and object, and finally of course any application layer on top of this that I could save as a stand-alone system. This seemed to fit with the evolving ideas in telecommunications standards providing portable levels or layers.

I was deeply influenced by my first exposure to Smalltalk. At that time my idea of an ideal environment was a Forth with Smalltalk-like object, graphic, and GUI extensions. It ultimately wanted to have a source browser assisted the user in tasks like metacompilation making them almost transparent. I did add a layer for a Smalltalk-like GUI toolkit to my Forth dictionary. It only adds a few K at most.

In the eighties I worked in telecommunications and I added appropriate telecom protocol layers into the system and support for new modems, and a scripting language layer, and of course scripts. I was also researching AI in Forth, implementing ideas from LISP examples and doing expert systems and neural nets and mixing them and building robots. In the software I added a layer for an inference engine for English language descriptions of rule sets and a layer for the rules. I wrote a learning email report and conversation engine AI program and had it running for a few months. My boss could not distinguish it from me. That was my idea of AI, smart enough to do my job for me and get paid at my salary while I took a vacation. A couple of years later I purchased a Forth kit with the first Forth chip and got personal friendly support from Charles Moore. My ideal language now needed to have neural and rule based background tasks intelligently cooking knowledge bases. I also studied parallel programming paradigms and worked on parallel programming extensions to Forth. [Fox, 1991] [Fox, 1995] My idealized environment became a simple and fast parallel Forth chip optimized for AI code and multimedia and an integrated version of all my Forth code. [Fox, 1993]

Now to jump ahead ten years, I found myself working with Charles Moore, the inventor of Forth, on our latest custom VLSI. It was be his fifth generation Forth chip, my first. I had worked with several of the earlier generations of chips but not contributed to or owned a license on the design as with the F21.

Working closely with him for a number of years changed most of my ideas about Forth. I thought I knew Forth by this time and was somewhat perplexed by what I saw Chuck do. I still could not get close to the results that he got, either in productivity or quality. I tried very hard to study and understand what he was doing different than the Forth that I had learned. I recall that the first thing that jumped out at me was that Chuck's code didn't have all those layers that my code had. There was the abstracted code on top and the hardware drivers on the bottom and very little in between. There was not a clear distinction in style at various levels as in my code where there were always someone else's code layered under mine that I could not or dared not change. Chuck's code looked abstracted and high level right from the bottom, the top used almost as many primitives as the bottom code.

Of course we had optimized the hardware to get an instruction set to minimize the software required to build an optimizing native code compiler for Forth. This new improved Forth virtual machine was now no longer a virtual layer in software, it was a layer of hardware. Chuck put most of his efforts into optimizing the hardware for speed and simplicity after the specification to support the improved virtual machine and simplified optimizing compiler that was about five years old now. This also eliminated the need for an assembler layer, our native code was Forth, Forth was the assembler. We couldn't have it much nicer than this.

I was working mostly with software for the actual chips while Chuck focused on the hardware, and using and refining his CAD software to aid him in this process. I observed that he applied the same style to his work on his PC based CAD software to the software intended for this own chips. I asked him about it and he said that he just ported to Pentium by mapping the ideas without adding any extra layers of abstraction. He said that his chips sort of enforced good Forth style but that on Pentium it takes a little more effort.

To jump forward another seven years, earlier this month Chuck made a comment in an interview about Forth in an Internet chat room. [Moore, 2002] He was talking about the notions of levels in the ANS Forth CATCH/THROW error handling method when he stated, “What I do is to mix the low and high-level code into an integrated whole; modify the low-level code as necessary for the application. The notion of levels of code, as in communication protocols, is wrong. There needn’t be so much code to make it necessary!”

It jumped out at me again. His approach is that there is a problem, constrained by hardware, and an ideal solution expressed by brevity, accuracy, reliability. In short quality. It’s the ideal solution and nothing but the ideal solution. Chuck’s code has no layers that are fixed. He wrote all of it and it is as easy for him to change any part of it as any other part of it. I knew that. No wonder he doesn’t need all those levels. No wonder he is so much more productive and writes so much better code. I had written essays about this before, he doesn’t use all those layers of abstractions. [Fox, 1998]

Later on in the chat session interview Chuck went on to say, “Portability is not important. Portability is not possible. Real applications are closely coupled to hardware. Change the platform and all the code changes. If it didn’t, you wouldn’t have changed the platform. To abstract the problem from the hardware requires massive software like Windows, that’s a permanent tax on all applications to save some one-time programming. Programmers should object to job-elimination concepts. Of course, jobs are actually multiplied to deal with the hyper-complex abstraction. And modern hardware has computers in the displays and disks. They’ve already made many interfaces portable. How many layers of portability are needed?” [Moore, 2002]

Which brings me to the heart of the essay. Jack Woehr quotes Chuck as saying, “Forth is more an approach than a specification for a programming language.” [Woehr, 1992] Chuck said that the reason for Forth was to liberate the programmer, to get them out from behind things that they couldn’t change, to let them realize that idealized solution at the cutting edge of the capabilities of any given the hardware, and to maximize their control over everything. [Moore, 1970]

There is something that is a near mirror image of this and completely reversed. The ANS Forth Programming Language specification IS a formal specification for a programming language. Instead of existing to avoid the inclusion of unwanted programming constraints on a solution it formalizes further layers of mechanisms to embrace and mask over these constraints, it adds to them. It mandates adding a few further layers of abstraction in the name ‘portability.’ It loses any notion that the return stack is actually a return stack. It looses any actual access to registers or even real memory. It looses any notion of how anything works below more layers of abstraction. It loses the real and useful behaviors of I/O devices behind layers of abstraction.

Some authors of large ANS Forth systems for popular environments have said that 99% of their work and code was to deal with problems of the constraints of the other abstracted layers in the systems into which they integrate these ANS Forths. They tell us that these constraints dwarf the standard Forth compilers themselves. This fact alone should account for Chuck’s claim that they will write at least ten times as much code as he will write to reach a solution. [Moore, 1999] I think Chuck is just being modest or hasn’t paid close attention to just how much code these people have to write. I think the evidence supports my claim that they are likely to write a hundred times as much code as Chuck will write for a given problem specification. Chuck has only claimed that ‘C’ programmers write a hundred times more code than he will write. [Moore, 1999]

The ANS Forth standard is at least one, maybe two orders of magnitude more complex than Mr. Moore’s approach to Forth. He says that code should be so simple that most type of errors simply can’t happen. In the late eighties and early nineties Chuck quit writing code in Forth and experimented with sourceless programming. His first versions of his VLSI CAD software, OKAD, were constructed without source using his tools in his OK operating system. Later he return to Forth programming and rewrote OKAD II under his new colorForth. [Moore, 2000] In the chat session Chuck was asked, “How did you come to the conclusion that Forth was too complex, and that sourceless programming was your next move?”

His reply was, “Maybe by reading the Forth Standard.” [Moore, 2002]

References:

[Fox, 1991] Fox, Jeffrey A., Forth-Linda Parallelizing Forth, FORML Conference Proceedings, 1991

[Fox, 1995] Fox, Jeffrey A., Distributed Shared Memory in Forth, Parallel Forth, Internet, 1995

[Fox, 1993] Fox, Jeffrey A. and Montvelishsky, Michael, F21 and F*F, Parallelizing Forth, FORML Conference Proceedings, 1993

[Moore, 2002] Moore, Charles H., Internet Chat, May 5, 2002

[Fox, 1998] Fox, Jeffrey A., Low Fat Computing, Internet, 1998

[Woehr, 1992] Woehr, Jack J., Seeing Forth, page 83, available from Offete Enterprises, San Mateo, CA, 1992

[Moore, 1970] Moore, Charles H. and Leach, Geoffrey C., FORTH - A Language for Interactive Computing (pdf) (html), Amsterdam NY: Mohasco Industries, Inc. (internal pub.) 1970.

[Moore, 1999] Moore, Charles H., 1x Forth, Charles Moore, Interview 4/13/99

[Moore, 1999] Moore, Charles H., Dispelling the User Illusion, SVFIG, 5/22/99

[Moore, 2000] Moore, Charles H., colorforth web site, Internet, 2000

Jeff Fox, 5/29/02

Library

All words are CMSIS-SVD compilant and designed to run on Mecrisp-Stellaris.

Words here will be for development, otherwise they should be in Projects or Examples

Maths Library

Feb 2018, Andrew Palm, a mathematician, has created a square root, trig, and inverse trig functions for 64bit fixed point (s31.32) numbers library that should run on any Cortex M. Tested here on a STM32F051.

DOWNLOAD:fixpt-math-lib.fs

Note

If you haven’t used s31.32 fixed point numbers before, the (usual floating number) decimal point is replaced with a COMMA

Examples

```
4,0 sqrt f.
  2,00000000000000000000000000000000  ok.
3000,02398 sqrt 6 f.n
  54,772474  ok.
90,0 sin f.
  1,00000000000000000000000000000000  ok.
0,5 asin f.
  29,99999998090788722038269042968750  ok.
0,0 cos f.
```

```
  1,00000000000000000000000000000000  ok.
0,5 acos f.
  60,00000001909211277961730957031250  ok.
90,0 tan f.
  0,00000000023283064365386962890625  ok.
45,0 atan f.
  88,72696999576874077320098876953125  ok.
90,0 deg2rad f.
  1,57079633558169007301330566406250  ok.
1,57079633558169007301330566406250 rad2deg 6 f.n
  90,000000  ok.
```

Read Cortex M0 chip ID

Reads the chip ID of a Cortex Mo

Tested on a Cortex M0, may not work with other STM32 models

Download: f0-device-electronic-signature.fs

Example

```
stm32id

Die xy coords: 2752589
Wafer Number: 14
Lot_num ascii encoded  [23:0]: 0x00413657  |  A 6 W
Lot_num ascii encoded [55:24]: 0x20323336  |    2 3 6
```

40 word dictionary pager

Dict just pages the Dictionary words in a 40 word high single column. Press enter or the space bar to continue.

Download: dict.fs

Example

```
dict

--- Mecrisp-Stellaris Dictionary, press any key to continue ---
stack-check.fs
sc
--- Mecrisp-Stellaris Core ---
2dup
2drop
2swap
2nip
2over
2tuck
...
```

Stack Check

Checks the Data Stack for leftover items, warns you if the stack > 0, then clears the Stack. Use it at the end of loops etc.

Download: stack-check.fs

Example

```
stackcheck

*** Stack fault: there were ** 3 ** items left on the stack
Stack cleared
ok.
```

Print Register Contents with Legends

In svd2forth-v3 (not yet ready for release) I’ve separated the *register legends* from the *memory map* to allow *customised legends for any MCU model*. These provide a lot more information with the autogenerated register prints. These are hand edited in a simple text file, so if you have a favorite STM32Fxx MCU, you will be able to make them yourself.

A “rcc.” snippet. You can easily see that all GPIO’s are enabled as is USART2, but not USART1.

```
rcc.

 RCC_AHBENR $005E0000
                   I   I I I I                         F
               T   O   O O O O                         L   S
               S   P   P P P P                     C   I   R
D
               C   F   D C B A                     R   T   A
M
               E   E   E E E E                     C   F   M
A
               N   N   N N N N                     E   E   E
E
              | 2|  | 2|  | 2| 1| 1| 1|            N   N   N
N

~~~~~~~~~~~~~| 4| ~| 2| ~| 0| 9| 8| 7| ~~~~~~~~~~~~~~~~~~~| 6| ~| 4| ~| 2| ~
| 0
 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

 RCC_APB2ENR $00000000
                   D       T T T   U
S
                   B       I I I   A   S T
Y
                   G       M M M   R   P I
S
                   M       1 1 1   T   I M   A
C
                   C       7 6 5   1   1 1   D
F
                   E       E E E   E   E E   C
G
                   N       N N N   N   N N   E
E
                 | 2|      | 1| 1| 1|  | 1|  | 1| 1|   N
N

~~~~~~~~~~~~~~~~~| 2| ~~~~~| 8| 7| 6| ~| 4| ~| 2| 1| ~| 9| ~~~~~~~~~~~~~~~
| 0
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

 RCC_APB1ENR $00020000
                             U
                   I I       A     S     W     T
   C D P           2 2       R     P     W     I       T
T T
   E A W           C C       T     I     D     M       I
I I
   C C R           2 1       2     2     G     1       M
M M
   E E E           E E       E     E     E     4       6
3 2
   N N N           N N       N     N     N     E       E
E E
  | 3| 2| 2|         | 2| 2|      | 1|    | 1|    | 1|     N       N
N N

~| 0| 9| 8| ~~~~~~~~~| 2| 1| ~~~~~| 7| ~~~| 4| ~~~| 1| ~~~| 8| ~~~~~| 4| ~~~| 1
| 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
```

GNU General Public License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc.
<http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license
for
software and other kinds of works.

  The licenses for most software and other practical works
are designed
to take away your freedom to share and change the works.  By
contrast,
the GNU General Public License is intended to guarantee your
freedom to
share and change all versions of a program--to make sure it
remains free
software for all its users.  We, the Free Software
Foundation, use the
GNU General Public License for most of our software; it
applies also to
any other work released this way by its authors.  You can
apply it to
your programs, too.

  When we speak of free software, we are referring to
freedom, not
price.  Our General Public Licenses are designed to make sure
that you
have the freedom to distribute copies of free software (and
charge for
them if you wish), that you receive source code or can get it
```

if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this

License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this

License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the

Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional

```
permissions
apply only to part of the Program, that part may be used
separately
under those permissions, but the entire Program remains
governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your
option
remove any additional permissions from that copy, or from any
part of
it.  (Additional permissions may be written to require their
own
removal in certain cases when you modify the work.)  You may
place
additional permissions on material, added by you to a covered
work,
for which you have or can give appropriate copyright
permission.

  Notwithstanding any other provision of this License, for
material you
add to a covered work, you may (if authorized by the
copyright holders of
that material) supplement the terms of this License with
terms:

    a) Disclaiming warranty or limiting liability differently
from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal
notices or
    author attributions in that material or in the
Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that
material, or
    requiring that modified versions of such material be
marked in
    reasonable ways as different from the original version;
or

    d) Limiting the use for publicity purposes of names of
```

```
licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use
of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of
that
    material by anyone who conveys the material (or modified
versions of
    it) with contractual assumptions of liability to the
recipient, for
    any liability that these contractual assumptions directly
impose on
    those licensors and authors.

  All other non-permissive additional terms are considered
"further
restrictions" within the meaning of section 10.  If the
Program as you
received it, or any part of it, contains a notice stating
that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document
contains
a further restriction but permits relicensing or conveying
under this
License, you may add to a covered work material governed by
the terms
of that license document, provided that the further
restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this
section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice
indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be
stated in the
form of a separately written license, or stated as
exceptions;
```

the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT

```
LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE
OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME
THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO
IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES
AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT
LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability
provided
above cannot be given local legal effect according to their
terms,
reviewing courts shall apply local law that most closely
approximates
an absolute waiver of all civil liability in connection with
the
Program, unless a warranty or assumption of liability
accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS
```

```
            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.
```

```
  If the program does terminal interaction, make it output a
short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for
details type `show w'.
    This is free software, and you are welcome to
redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show
the appropriate
parts of the General Public License.  Of course, your
program's commands
might be different; for a GUI interface, you would use an
"about box".

  You should also get your employer (if you work as a
programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if
necessary.
For more information on this, and how to apply and follow the
GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit
incorporating your program
into proprietary programs.  If your program is a subroutine
library, you
may consider it more useful to permit linking proprietary
applications with
the library.  If this is what you want to do, use the GNU
Lesser General
Public License instead of this License.  But first, please
read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

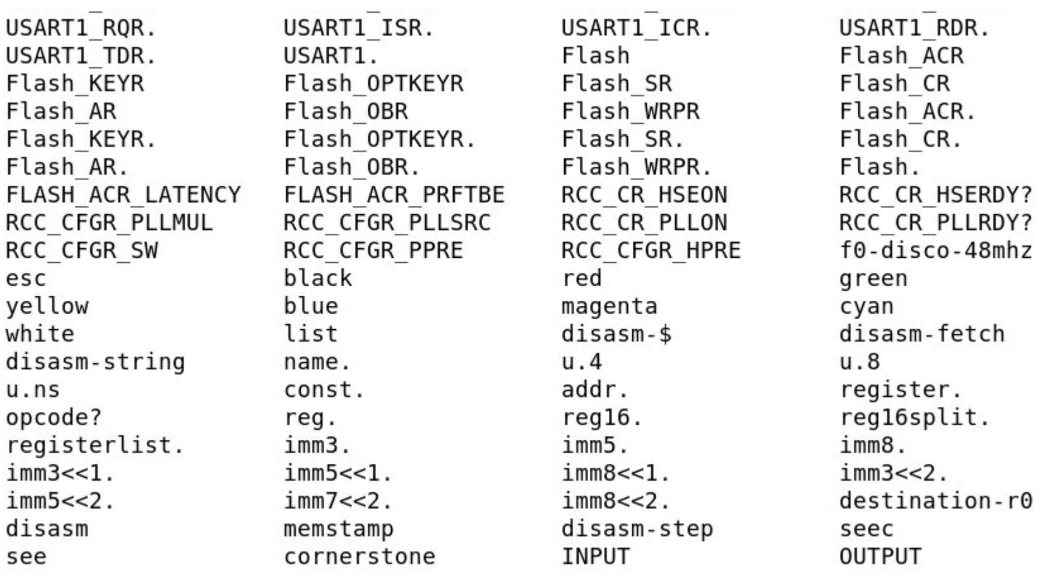
Words4

words4 sample output.

Note

This is a non standard Wordlist using Words generated by SVD2FORTH.

```
ud.                 d.                  u.                  .
evaluate            interpret           hook-quit           quit
eint?               eint                dint                ipsr
nop                 unhandled           reset               irq-systick
irq-fault           irq-collection      irq-exti0_1         irq-exti2_3
irq-exti4_15        irq-tsc             irq-dma_ch1         irq-dma_ch2_3
irq-dma_ch4_5       irq-adc             irq-tim1_up         irq-tim1_cc
irq-tim2            irq-tim3            irq-tim6_dac        irq-tim14
irq-tim15           irq-tim16           irq-tim17           irq-i2c1
irq-i2c2            irq-spi1            irq-spi2            irq-usart1
irq-usart2          irq-cec_can         --- Flash Dictionary ---
b32loop.            b16loop-a.          b16loop.            b8loop.
4bl.                4bh.                2b.                 1b.
15b.                b32sloop.           bin.                u.4
u.2                 dump16              dump                TEMPLATE
TEMPLATE..          TEMPLATE.           RCC_CR..            RCC_CFGR..
RCC_AHBENR..        RCC_APB2ENR..       RCC_APB1ENR..       RCC_CSR..
RCC_CFGR2..         RCC_CFGR3..         RCC_CR2..           COMP_CSR..
GPIOC               GPIOC_MODER         GPIOC_OTYPER        GPIOC_OSPEEDR
GPIOC_PUPDR         GPIOC_IDR           GPIOC_ODR           GPIOC_BSRR
GPIOC_LCKR          GPIOC_AFRL          GPIOC_AFRH          GPIOC_BRR
GPIOC_MODER.        GPIOC_OTYPER.       GPIOC_OSPEEDR.      GPIOC_PUPDR.
GPIOC_IDR.          GPIOC_ODR.          GPIOC_BSRR.         GPIOC_LCKR.
GPIOC_AFRL.         GPIOC_AFRH.         GPIOC_BRR.          GPIOC.
GPIOA               GPIOA_MODER         GPIOA_OTYPER        GPIOA_OSPEEDR
GPIOA_PUPDR         GPIOA_IDR           GPIOA_ODR           GPIOA_BSRR
GPIOA_LCKR          GPIOA_AFRL          GPIOA_AFRH          GPIOA_BRR
GPIOA_MODER.        GPIOA_OTYPER.       GPIOA_OSPEEDR.      GPIOA_PUPDR.
GPIOA_IDR.          GPIOA_ODR.          GPIOA_BSRR.         GPIOA_LCKR.
GPIOA_AFRL.         GPIOA_AFRH.         GPIOA_BRR.          GPIOA.
RCC                 RCC_CR              RCC_CFGR            RCC_CIR
RCC_APB2RSTR        RCC_APB1RSTR        RCC_AHBENR          RCC_APB2ENR
RCC_APB1ENR         RCC_BDCR            RCC_CSR             RCC_AHBRSTR
RCC_CFGR2           RCC_CFGR3           RCC_CR2             RCC_CR.
RCC_CFGR.           RCC_CIR.            RCC_APB2RSTR.       RCC_APB1RSTR.
RCC_AHBENR.         RCC_APB2ENR.        RCC_APB1ENR.        RCC_BDCR.
RCC_CSR.            RCC_AHBRSTR.        RCC_CFGR2.          RCC_CFGR3.
RCC_CR2.            RCC.                USART1              USART1_CR1
USART1_CR2          USART1_CR3          USART1_BRR          USART1_GTPR
USART1_RTOR         USART1_RQR          USART1_ISR          USART1_ICR
USART1_RDR          USART1_TDR          USART1_CR1.         USART1_CR2.
USART1_CR3.         USART1_BRR.         USART1_GTPR.        USART1_RTOR.
```



```
USART1_RQR.         USART1_ISR.         USART1_ICR.         USART1_RDR.
USART1_TDR.         USART1.             Flash               Flash_ACR
Flash_KEYR          Flash_OPTKEYR       Flash_SR            Flash_CR
Flash_AR            Flash_OBR           Flash_WRPR          Flash_ACR.
Flash_KEYR.         Flash_OPTKEYR.      Flash_SR.           Flash_CR.
Flash_AR.           Flash_OBR.          Flash_WRPR.         Flash.
FLASH_ACR_LATENCY   FLASH_ACR_PRFTBE    RCC_CR_HSEON        RCC_CR_HSERDY?
RCC_CFGR_PLLMUL     RCC_CFGR_PLLSRC     RCC_CR_PLLON        RCC_CR_PLLRDY?
RCC_CFGR_SW         RCC_CFGR_PPRE       RCC_CFGR_HPRE       f0-disco-48mhz
esc                 black               red                 green
yellow              blue                magenta             cyan
white               list                disasm-$            disasm-fetch
disasm-string       name.               u.4                 u.8
u.ns                const.              addr.               register.
opcode?             reg.                reg16.              reg16split.
registerlist.       imm3.               imm5.               imm8.
imm3<<1.            imm5<<1.            imm8<<1.            imm3<<2.
imm5<<2.            imm7<<2.            imm8<<2.            destination-r0
disasm              memstamp            disasm-step         seec
see                 cornerstone         INPUT               OUTPUT
```

Viewing The Mecrisp-Stellaris Dictionary

To view the Mecrisp-Stellaris Dictionary one can use the following Words. Each one has a different format:

Words

words sample output.

```
words

...
Address: 0000CDB8 Link: 0000CDDC Flags: 00000030 Code:
0000CDC2 Name: str
Address: 0000CDDC Link: 0000CE04 Flags: 00000030 Code:
0000CDE8 Name: strb
Address: 0000CE04 Link: 0000CE2C Flags: 00000030 Code:
0000CE10 Name: strh
Address: 0000CE2C Link: 0000CE50 Flags: 00000030 Code:
0000CE36 Name: ldr
Address: 0000CE50 Link: 0000CE78 Flags: 00000030 Code:
0000CE5C Name: ldrb
Address: 0000CE78 Link: 0000CEA0 Flags: 00000030 Code:
```

```
0000CE84 Name: ldrh
...
```

List

list sample output. “list” is part of the /common/dissasembler-m0.txt and /common/dissasembler-m3.txt files.

```
list

...
+ - 1- 1+ 2- 2+ cell+ negate not shr shl 2* cells 2/ abs
u/mod /mod mod / even base binary decimal hex hook-emit
hook-key hook-emit? hook-key? hook-pause emit key emit? key?
pause serial-emit serial-key serial-emit? serial-key?
cexpect accept tib >in current-source setsource source query
compare cr bl space spaces [char] char ( \ ." c" s"
count ctype type hex. h.s u.s .s words unused
registerliteral, call, literal, create does> <builds ['] '
postpone
inline, ret, exit recurse state ] [ : ; execute immediate
inline compileonly 0-foldable 1-foldable 2-foldable
3-foldable 4-foldable 5-foldable 6-foldable 7-foldable
constant 2constant smudge setflags aligned align h, , ><,
string, allot forgetram compiletoram? compiletoram
compiletoflash (create) variable 2variable nvariable buffer:
dictionarystart dictionarynext skipstring find hook-find
(find) cjump, jump, here flashvar-here then else if ahead
repeat while until again begin k j i leave unloop +loop loop
do ?do case ?of of endof endcase token parse digit
number .digit hold hold< sign #> f#S f# #S # <# f. f.n ud. d.
u. . evaluate interpret hook-quit quit eint? eint
dint ipsr nop unhandled reset irq-systick irq-fault irq-
collection irq-exti0_1 irq-exti2_3 irq-exti4_15 irq-tsc
...
```

List Source

DOWNLOAD list.fs

```
\ Copyright (C) 2013  Matthias Koch
\ This is free software under GNU General Public License v3.


: list ( -- )
  cr
  dictionarystart
  begin
    dup 6 + ctype space
    dictionarynext
  until
  drop
;
```

Words4

words4 sample output.

```
words4

...
nop                 unhandled           reset
irq-systick
irq-fault           irq-collection      irq-exti0_1
irq-exti2_3
irq-exti4_15        irq-tsc             irq-dma_ch1
irq-dma_ch2_3
irq-dma_ch4_5       irq-adc             irq-tim1_up
irq-tim1_cc
irq-tim2            irq-tim3            irq-tim6_dac
irq-tim14
irq-tim15           irq-tim16           irq-tim17
irq-i2c1
irq-i2c2            irq-spi1            irq-spi2
irq-usart1
irq-usart2          irq-cec_can         --- Flash Dictionary
---
b32loop.            b16loop-a.          b16loop.
b8loop.
4bl.                4bh.                2b.
1b.
15b.                b32sloop.           bin.
u.4
```

```
u.2                 dump16              dump
TEMPLATE
...
```

Words4 Creation

I wanted a neat, four column lister that printed like the listing above, with 20 character wide columns, but also handled Words greater than 20 characters. My “words4” source (based on word.fs above) is below.

Words4 Source

DOWNLOAD words4.fs

```
\ Program Name: words4.fs
\ This program may require preloaded support files
\ Date: Thu 25 Jul 2019 23:27:35 AEST
\ Copyright 2018  t.porter <terry@tjporter.com.au>, licensed
under the GPL
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ 140 bytes

: words4 ( -- ) cr  \ A columnar Word list printer. Width =
20 characters, handles overlength Words neatly
   0                                \ column counter
   dictionarystart
      begin
         dup 6 + dup
         ctype                      \ dup before 6 is for
dictionarynext input
         count nip                  \ get number of
characters in the word and drop the address of the word
             20 swap - dup 0 > if   \ if Word is less than 20
chars
                   spaces swap      \ pad with spaces to
equal 20 chars
                   else drop cr     \ issue immediate
carriage return and drop negative number
                   nip -1           \ and reset to column -1
                   then
                      dup 3 = if 3 - cr \ if at 4th column,
zero column counter
                      else 1 +
                      then
         swap
         dictionarynext             \   ( a-addr - - a-addr
flag )
      until
   2drop
;
```

Key Words Used

```
dictionarystart        ( - - a-addr )  Current entry point
for dictionary search
dictionarynext         ( a-addr - - a-addr flag )      Scans
dictionary chain and returns true if end is reached.
ctype  ( cstr-addr - - )       Prints a counted string.
count  ( cstr-addr - - c-addr length )         Convert
counted string into addr-length string
```

Words4 Operation

Words4 utilises the default “dictionarystart and dictionarynext” Words to iterate thru each Word name to the end of the dictionary. Each Word name is printed with “ctype” and the number of characters is determined using “count”.

As each column is 20 characters wide, any Word of less than 20 characters is ‘padded’ out with spaces so that it is exactly 20 characters in length.

A column counter emits a carriage return after four Word names have been printed.

Words greater than 20 characters end the column sequence after they are printed, and normal operation continues on the next line as illustrated by the “— Flash Dictionary —” Word above.

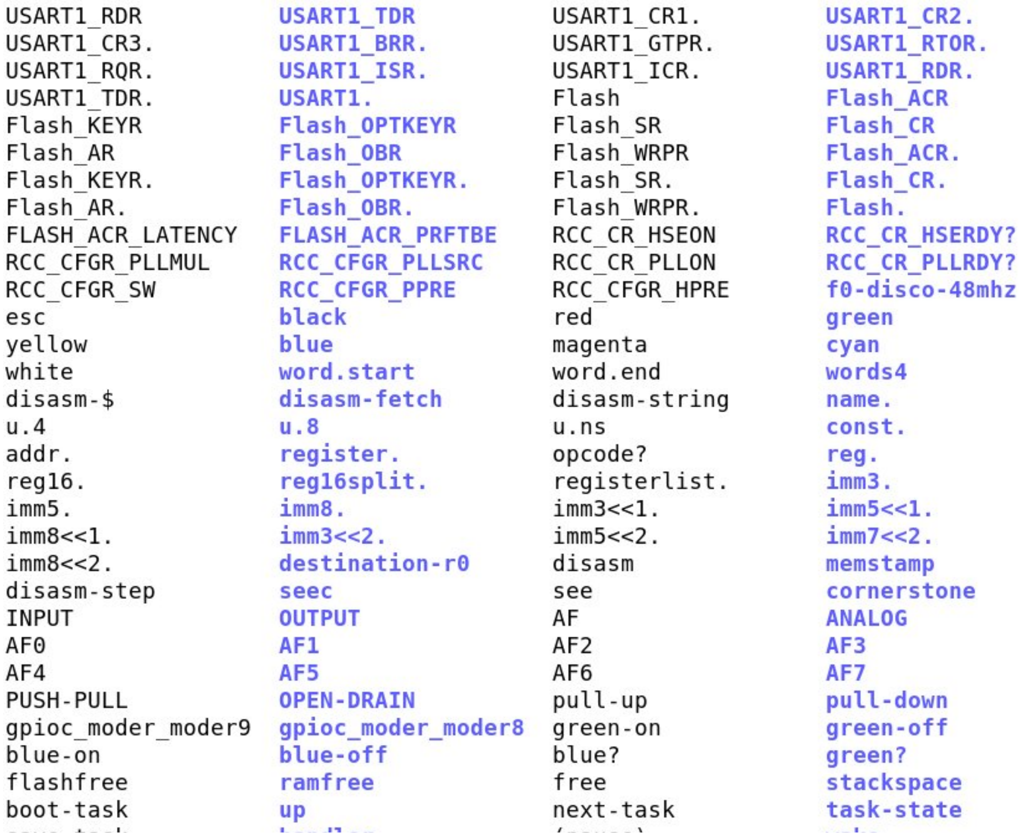
This code maintains 2 items on the data stack throughout its operation, they are the address of each Word and the column number, so to better understand how it works one should view the data stack after each line with “.s”. Reading at my initial ‘proof of concept’ code below (list4b) may assist the understanding of what is being done.

Coloured Columns

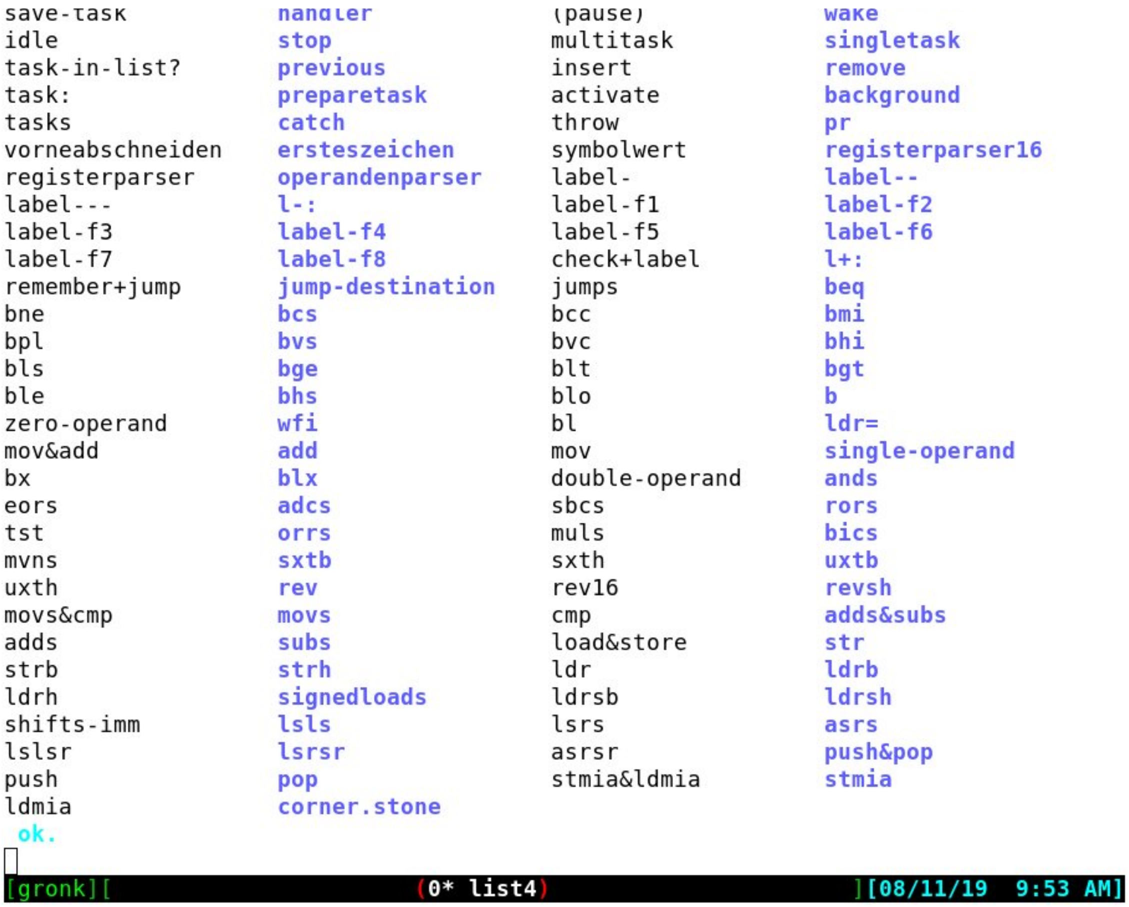
This version adds colour to the list.

Note

requires colour-term.fs



```
USART1_RDR          USART1_TDR          USART1_CR1.         USART1_CR2.
USART1_CR3.         USART1_BRR.         USART1_GTPR.        USART1_RTOR.
USART1_RQR.         USART1_ISR.         USART1_ICR.         USART1_RDR.
USART1_TDR.         USART1.             Flash               Flash_ACR
Flash_KEYR          Flash_OPTKEYR       Flash_SR            Flash_CR
Flash_AR            Flash_OBR           Flash_WRPR          Flash_ACR.
Flash_KEYR.         Flash_OPTKEYR.      Flash_SR.           Flash_CR.
Flash_AR.           Flash_OBR.          Flash_WRPR.         Flash.
FLASH_ACR_LATENCY   FLASH_ACR_PRFTBE    RCC_CR_HSEON        RCC_CR_HSERDY?
RCC_CFGR_PLLMUL     RCC_CFGR_PLLSRC     RCC_CR_PLLON        RCC_CR_PLLRDY?
RCC_CFGR_SW         RCC_CFGR_PPRE       RCC_CFGR_HPRE       f0-disco-48mhz
esc                 black               red                 green
yellow              blue                magenta             cyan
white               word.start          word.end            words4
disasm-$            disasm-fetch        disasm-string       name.
u.4                 u.8                 u.ns                const.
addr.               register.           opcode?             reg.
reg16.              reg16split.         registerlist.       imm3.
imm5.               imm8.               imm3<<1.            imm5<<1.
imm8<<1.            imm3<<2.            imm5<<2.            imm7<<2.
imm8<<2.            destination-r0      disasm              memstamp
disasm-step         seec                see                 cornerstone
INPUT               OUTPUT              AF                  ANALOG
AF0                 AF1                 AF2                 AF3
AF4                 AF5                 AF6                 AF7
PUSH-PULL           OPEN-DRAIN          pull-up             pull-down
gpioc_moder_moder9  gpioc_moder_moder8  green-on            green-off
blue-on             blue-off            blue?               green?
flashfree           ramfree             free                stackspace
boot-task           up                  next-task           task-state
```



```
save-task           handler             (pause)             wake
idle                stop                multitask           singletask
task-in-list?       previous            insert              remove
task:               preparetask         activate            background
tasks               catch               throw               pr
vorneabschneiden    ersteszeichen       symbolwert          registerparser16
registerparser      operandenparser     label-              label--
label---            l-:                 label-f1            label-f2
label-f3            label-f4            label-f5            label-f6
label-f7            label-f8            check+label         l+:
remember+jump       jump-destination    jumps               beq
bne                 bcs                 bcc                 bmi
bpl                 bvs                 bvc                 bhi
bls                 bge                 blt                 bgt
ble                 bhs                 blo                 b
zero-operand        wfi                 bl                  ldr=
mov&add             add                 mov                 single-operand
bx                  blx                 double-operand      ands
eors                adcs                sbcs                rors
tst                 orrs                muls                bics
mvns                sxtb                sxth                uxtb
uxth                rev                 rev16               revsh
movs&cmp            movs                cmp                 adds&subs
adds                subs                load&store          str
strb                strh                ldr                 ldrb
ldrh                signedloads         ldrsb               ldrsh
shifts-imm          lsls                lsrs                asrs
lslsr               lsrsr               asrsr               push&pop
push                pop                 stmia&ldmia         stmia
ldmia               corner.stone
 ok.

[gronk][                      (0* list4)                      ][08/11/19  9:53 AM]
```

Words4c

Note

This is for a Terminal with a WHITE background, change “black” Words for another colour below if you use a BLACK background

```
: words4c ( -- ) cr                \ A coloured columnar Word
list printer. Width = 20 characters.
   0                               \ column counter
   dictionarystart
      begin
        dup 6 + dup
        3 pick                     \ copy the column counter
value
           case                    \ NOTE: colours here are
```

```
for a WHITE terminal BACKGROUND !
              0 of black  endof
              1 of blue endof
              2 of black  endof
              3 of blue endof
            endcase
        ctype                      \ print word
        white
        count nip                  \ get number of characters
in the word and drop the address of the word
            20 swap - dup 0 > if   \ if Word is less than 20
chars
                  spaces swap      \ pad with spaces to equal
20 chars
                  else drop cr     \ issue immediate carriage
return and drop negative number
                  nip -1           \ and reset to column -1
                  then
                     dup 3 = if 3 - cr \ if at 4th column,
zero column counter
                     else 1 +
                     then
        swap
        dictionarynext             \   ( a-addr - - a-addr
flag )
      until
   2drop
;
```

Machine and Assembly Code

Note

Mecrisp-Stellaris 2.4.2-RA used here

See also

Interactive Assembly Language Words and Accurate Blocking Delay

Ra Register Allocator Usage

RA	Register Alocator Register	USAGE
r0	Free scratch register	Saved on interrupt entry by hardware
r1	Free scratch register	Saved on interrupt entry by hardware
r2	Free scratch register	Saved on interrupt entry by hardware

RA	Register Alocator Register	USAGE
r3	Free scratch register	Saved on interrupt entry by hardware
r4	Inner loop count	Needs Push and Pop when used otherwise
r5	Inner loop limit	Needs Push and Pop when used otherwise
r6	TOS: Top-Of-Stack	Stack design is interrupt safe
r7	PSP: Parameter Stack Pointer	Stack design is interrupt safe
r8	Unused	
r9	Unused	
r10	Unused	
r11	Unused	

RA	Register Alocator Register	USAGE
r12	Unused	Saved on interrupt entry by hardware
r13	SP Return Stack Pointer	
r14	LR Link Register	
r15	PC Program Counter, always odd	

Executing Assembly/Machine Code

Executing and or Inlining Assembly/Machine code is fairly easily handled by Mecrisp-Stellaris and the following workflow example shows one way of doing it.

Methodology

- Write your assemby code
- Assemble your code
- Paste the machine code into a Word
- Dissasemble the Word
- Test the Word

Warning

- You are free to change R0, R1, R2 and R3
- R4 and R5 need to pushed and popped if used.
- R6 is the Top of Stack (TOS) on entry and exit. If there are more than one result, they have to be pushed onto to the stack.
- Stack pointer R7 must be used in an interrupt-safe way.
- Subroutines cannot be used
- When you use "[" Mecrisp-Stellaris assumes "everything might happen" and frames your code with push { lr} / pop { pc} opcodes

Notes

1. When writing to the Stack, the Stack Index Register (R7) must be decremented by 4 first and the existing Stack contents saved (pushed down one level) to the Stack.

::

> subs r7 #4 R7 = R7 - 4 R7 is Data Stack Pointer, so decrement it by 4 (8 bit bytes) str r6 r7 #0 Save old contents of R6 at new memory location specified by R7

Example 1: 2plus (adds 2 to any number)

Write 2plus.s assemby code

I'm only interested in the assembled code after "start:" because Mecrisp-Stellaris has its own Vector Table and Initialisation Code.

```
// Minimal assembler example for stm32f051 to be used in a
Machine Code Word "2plus"
// Requires svd2as
// Board: stm32f0 Discovery
// 1 June 2018  Copyright <terry@tjporter.com.au> Released
under the GPL
//

.cpu cortex-m0                                       // Tell
the assembler what model of CortexM this is for
.thumb                                               //
Cortex micros only understand thumb(2) code
.text                                                // what
follows is code
.include "bitposn-equates.s"                         // a
list of ".equ  BIT0,    0x00000001", ".equ  BIT1,
0x00000002" etc
.include "STM32F0xx-tp1.svd.s"                       //
Created by svd2as
.syntax unified // use newer style instructions
Vector_Table:  .word     0x20000000                  // stack
pointer value when stack is empty 0x20000000
ResetVector:    .word     start + 1                  // Reset
Handler

start:
// Input parameter will be saved in r6 (TOS)
        movs r1, #2                          // move 2 into
r1
        adds r2, r6, r1                       // add with
carry, r1 + r6 and place result into r2. Note: r6 is TOS on
entry
        movs r6, r2                           // move the
contents of r2 into r6. Note: the result is placed back onto
TOS via r6 on exit
```

Assemble 2plus.s

This text is from 2plus.list produced by this Makefile.

```
2plus.elf:     file format elf32-littlearm

Disassembly of section .text:

```

```
00000000 <Vector_Table>:
   0:  20000000        andcs   r0, r0, r0

00000004 <ResetVector>:
   4:  00000009        andeq   r0, r0, r9

00000008 <start>:
   8:  2102            movs    r1, #2
   a:  1872            adds    r2, r6, r1
   c:  0016            movs    r6, r2
```

Paste The Machine Code Into 2plus Word

Add comments for readability

```
: 2plus ( u -- 2+ )
[           \ *execute* mode
$2102 h,    \ movs      r1, #2
$1872 h,    \ adds      r2, r6, r1
$0016 h,    \ movs      r6, r2
]           \ *compile* mode
;
```

Dissasemble 2plus

Compare this to the 2plus Word ? The Machine Code is unchanged. Mecrisp-Stellaris has framed the code with push { lr} / pop { pc} opcodes

```
see 2plus

200003AC: B500  push { lr }
200003AE: 2102  movs r1 #2
200003B0: 1872  adds r2 r6 r1
200003B2: 0016  lsls r6 r2 #0
200003B4: BD00  pop { pc }
ok.
```

Test 2plus

```
2 2plus . 4  ok.
```

2plus.s Makefile

```
ARMGNU?=arm-none-eabi

COPS = -Wall  -Os -nostdlib -nostartfiles -ffreestanding -save-temps
AOPS = --warn --fatal-warnings

all : 2plus.bin

2plus.o : 2plus.s
	$(ARMGNU)-as -mthumb -a=2plus.lst -g --gstabs+ 2plus.s -o 2plus.o

2plus.bin : memmap 2plus.o
	$(ARMGNU)-ld -Ttext 0x0 -o 2plus.elf -T memmap 2plus.o
	$(ARMGNU)-objdump -D 2plus.elf > 2plus.list
	$(ARMGNU)-objcopy 2plus.elf 2plus.bin -O binary

erase:
	st-flash erase

flash:
	st-flash write 2plus.bin 0x08000000

clean:
	rm -f *.bin
	rm -f *.o
	rm -f *.elf
	rm -f *.list
	rm -f *.lst
```

Download all the Files from the 2plus project

Example 2, more complex code, A blinky

This blinks the Blue led on a STM32F0 Discovery Board.

Methodology

- Write your assemby code
- Assemble your code
- Paste the machine code into a Word
- Dissassemble the Word
- Test the Word

Blinky Assembly Code

- Requires svd2gas to generate the hardware equate statements

```
// Minimal assembler blinky example for stm32f051
// Requires svd2as
// Board: stm32f0 Discovery
// Blue LED is on GPIOC bit 8
// 8th June 2018  Copyright <terry@tjporter.com.au> Released
under the GPL

.cpu cortex-m0                                    // Tell the
assembler what model of CortexM this is for
.thumb                                            // Cortex
micros only understand thumb(2) code
.text                                             // what
follows is code
.include "bitposn-equates.s"                      // a list
of ".equ  BIT0,    0x00000001", ".equ  BIT1," etc
.include "STM32F0xx.svd.s"                        //
Compliments of svd2gas
.syntax unified // use newer style instructions
Vector_Table:  .word     0x20000000               // stack
pointer value when stack is empty 0x20000000
ResetVector:    .word     start + 1               // Reset
Handler

start:
        ldr r1, = RCC_AHBENR
        ldr r2, = RCC_IOPCEN                      // Turn on
clock for GPIOC
        str r2, [r1]

moder:
        ldr r1, = GPIOC_MODER
        movs r2, 0b01                             // Set
GPIOC-8 to output
        lsls r2, GPIOC_MODER8_Shift               // BitWidth
2
        str r2, [r1]


led_on:                                           // Main
Loop
        ldr r1, = GPIOC_ODR
        ldr r2, = BLUE_LED_ON                     // Turn LED
on
        str r2, [r1]
        ldr r0, = DELAY                           // delay
delay_1: subs r0,r0,#1
        bne delay_1

led_off:
        ldr r1, = GPIOC_ODR
        ldr r2, = BLUE_LED_OFF                    // Turn LED
off
        str r2, [r1]
        ldr r0, = DELAY                           // Delay
delay_2: subs r0,r0,#1
        bne delay_2

        b led_on                                  // Jump to
main_loop

constants:
        .equ DELAY, 0xfffff
        .equ BLUE_LED_ON, GPIOC_ODR8
        .equ BLUE_LED_OFF, 0x0
```

Blinky.list Output

```
blinky.elf:     file format elf32-littlearm

Disassembly of section .text:

00000000 <Vector_Table>:
   0:  20000000        andcs   r0, r0, r0

00000004 <ResetVector>:
   4:  00000009        andeq   r0, r0, r9

00000008 <start>:
   8:  4909            ldr     r1, [pc, #36]   ; (30
<constants>)
   a:  4a0a            ldr     r2, [pc, #40]   ; (34
<constants+0x4>)
   c:  600a            str     r2, [r1, #0]

0000000e <moder>:
   e:  490a            ldr     r1, [pc, #40]   ; (38
<constants+0x8>)
  10:  2201            movs    r2, #1
  12:  0412            lsls    r2, r2, #16
  14:  600a            str     r2, [r1, #0]

00000016 <led_on>:
  16:  4909            ldr     r1, [pc, #36]   ; (3c
<constants+0xc>)
  18:  4a09            ldr     r2, [pc, #36]   ; (40 <BIT6>)
  1a:  600a            str     r2, [r1, #0]
  1c:  4809            ldr     r0, [pc, #36]   ; (44
<BIT6+0x4>)

0000001e <delay_1>:
  1e:  3801            subs    r0, #1
  20:  d1fd            bne.n   1e <delay_1>

00000022 <led_off>:
  22:  4906            ldr     r1, [pc, #24]   ; (3c
<constants+0xc>)
  24:  4a08            ldr     r2, [pc, #32]   ; (48
<BIT6+0x8>)
  26:  600a            str     r2, [r1, #0]
  28:  4806            ldr     r0, [pc, #24]   ; (44
<BIT6+0x4>)

0000002a <delay_2>:
  2a:  3801            subs    r0, #1
  2c:  d1fd            bne.n   2a <delay_2>
  2e:  e7f2            b.n     16 <led_on>

00000030 <constants>:
  30:  40021014        andmi   r1, r2, r4, lsl r0
  34:  00080000        andeq   r0, r8, r0
  38:  48000800        stmdami r0, {fp}
  3c:  48000814        stmdami r0, {r2, r4, fp}
  40:  00000100        andeq   r0, r0, r0, lsl #2
  44:  000fffff        strdeq  pc, [pc], -pc   ;
<UNPREDICTABLE>
  48:  00000000        andeq   r0, r0, r0
```

Paste Blinky machine code into a Word

Note

1) Although the Mecrisp Stellaris Kernel enables all the GPIOs, GPIOC-8 (the blue led) is enabled again here, which is totally redundant but doesn’t affect anything.

```
: blinky ( -- )
[
$4909 h,    \  ldr     r1, [pc, #36]   ; (30 <constants>)
$4a0a h,    \  ldr     r2, [pc, #40]   ; (34 <constants+0x4>)
$600a h,    \  str     r2, [r1, #0]
$490a h,    \  ldr     r1, [pc, #40]   ; (38 <constants+0x8>)
$2201 h,    \  movs    r2, #1
$0412 h,    \  lsls    r2, r2, #16
$600a h,    \  str     r2, [r1, #0]
$4909 h,    \  ldr     r1, [pc, #36]   ; (3c <constants+0xc>)
$4a09 h,    \  ldr     r2, [pc, #36]   ; (40 <BIT6>)
$600a h,    \  str     r2, [r1, #0]
$4809 h,    \  ldr     r0, [pc, #36]   ; (44 <BIT6+0x4>)
$3801 h,    \  subs    r0, #1
$d1fd h,    \  bne.n   1e <GPIOC_MODER15_Shift>
$4906 h,    \  ldr     r1, [pc, #24]   ; (3c <constants+0xc>)
$4a08 h,    \  ldr     r2, [pc, #32]   ; (48 <BIT6+0x8>)
$600a h,    \  str     r2, [r1, #0]
$4806 h,    \  ldr     r0, [pc, #24]   ; (44 <BIT6+0x4>)
$3801 h,    \  subs    r0, #1
$d1fd h,    \  bne.n   2a <BIT5+0xa>
$e7f2 h,    \  b.n     16 <GPIOC_MODER11_Shift>
            \ <constants>:
$40021014 , \  andmi   r1, r2, r4, lsl r0
$00080000 , \  andeq   r0, r8, r0
$48000800 , \  stmdami r0, {fp}
$48000814 , \  stmdami r0, {r2, r4, fp}
$00000100 , \  andeq   r0, r0, r0, lsl #2
$000fffff , \  strdeq  pc, [pc], -pc   ; <UNPREDICTABLE>
$00000000 , \  andeq   r0, r0, r0
]
;
```

Dissasemble the Blinky Word

This Blinky is 68 bytes long, not including the PUSH and POP added by Mecrisp-Stellaris.

```
see blinky
200003AE: B500  push { lr }
200003B0: 4909  ldr r1 [ pc #24 ]  Literal 200003D8: 40021014
200003B2: 4A0A  ldr r2 [ pc #28 ]  Literal 200003DC: 00080000
200003B4: 600A  str r2 [ r1 #0 ]
200003B6: 490A  ldr r1 [ pc #28 ]  Literal 200003E0: 48000800
200003B8: 2201  movs r2 #1
200003BA: 0412  lsls r2 r2 #10
200003BC: 600A  str r2 [ r1 #0 ]
200003BE: 4909  ldr r1 [ pc #24 ]  Literal 200003E4: 48000814
200003C0: 4A09  ldr r2 [ pc #24 ]  Literal 200003E8: 00000100
200003C2: 600A  str r2 [ r1 #0 ]
200003C4: 4809  ldr r0 [ pc #24 ]  Literal 200003EC: 000FFFFF
200003C6: 3801  subs r0 #1
200003C8: D1FD  bne 200003C6
200003CA: 4906  ldr r1 [ pc #18 ]  Literal 200003E4: 48000814
200003CC: 4A08  ldr r2 [ pc #20 ]  Literal 200003F0: 00000000
200003CE: 600A  str r2 [ r1 #0 ]
200003D0: 4806  ldr r0 [ pc #18 ]  Literal 200003EC: 000FFFFF
200003D2: 3801  subs r0 #1
200003D4: D1FD  bne 200003D2
200003D6: E7F2  b 200003BE
200003D8: 1014  asrs r4 r2 #0
200003DA: 4002  ands r2 r0
200003DC: 0000  lsls r0 r0 #0
200003DE: 0008  lsls r0 r1 #0
200003E0: 0800  lsrs r0 r0 #0
200003E2: 4800  ldr r0 [ pc #0 ]  Literal 200003E4: 48000814
200003E4: 0814  lsrs r4 r2 #0
200003E6: 4800  ldr r0 [ pc #0 ]  Literal 200003E8: 00000100
200003E8: 0100  lsls r0 r0 #4
200003EA: 0000  lsls r0 r0 #0
```

```
200003EC: FFFF
200003EE: 000F  lsls r7 r1 #0
200003F0: 0000  lsls r0 r0 #0
200003F2: 0000  lsls r0 r0 #0
200003F4: BD00  pop { pc }
ok.
```

Test the Blinky Word

```
blinky
```

Note

It works fine, the blue led blinks, but it’s a closed loop and requires resetting the board to stop blinking.

Comparison to a Blinky written in Forth

How will the assembly Blinky compare in code size compared to a Native Forth Blinky ?

Note

Answer: the Forth version is exactly twice the size in bytes.

Native Forth Blinky

Certainly *much* easier to write, only 8 lines of code. I pasted the Bitfields complete with memmory-mappings from svd2forth-v4 (not ready for release yet), making this a simple task.

This ran correctly first try. The code is not the most readable as I tried to emulate the assembly code layout as closely as possible.

```
: blinky ( -- )                       \ enable GPIOC-8 as a
OUTPUT here for fair comparison to blinky-asm
   %1  19  lshift $40021014  bis!     \ RCC_AHBENR_IOPCEN;
I/O port C clock enable
   %01 16  lshift $48000800  bis!     \ GPIOC_MODER_MODER8
set as OUTPUT
   begin                              \ endless loop
      %1  8  lshift $48000818   bis!  \ GPIOC_BSRR_BS8, turn
on blue led
      $fffff 0 do loop                \ delay $fffff
      %1  24  lshift $48000818  bis!  \ GPIOC_BSRR_BR8, turn
off blue led
      $fffff 0 do loop                \ delay $fffff
   0 until
 ;
```

Native Forth Blinky Assembly Listing

This is 136 Bytes long, making it exactly double the size of the assembly Blinky (68 Bytes).

```
see blinky
2000040A: 2080  movs r0 #80
2000040C: 0340  lsls r0 r0 #D
2000040E: 3084  adds r0 #84
20000410: 0280  lsls r0 r0 #A
20000412: 6943  ldr r3 [ r0 #14 ]
20000414: 2280  movs r2 #80
20000416: 0312  lsls r2 r2 #C
20000418: 4313  orrs r3 r2
2000041A: 6143  str r3 [ r0 #14 ]
2000041C: 2090  movs r0 #90
2000041E: 04C0  lsls r0 r0 #13
20000420: 3080  adds r0 #80
20000422: 0100  lsls r0 r0 #4
20000424: 6803  ldr r3 [ r0 #0 ]
20000426: 2280  movs r2 #80
20000428: 0252  lsls r2 r2 #9
2000042A: 4313  orrs r3 r2
```

```
2000042C: 6003  str r3 [ r0 #0 ]
2000042E: B500  push { lr }
20000430: 2090  movs r0 #90
20000432: 04C0  lsls r0 r0 #13
20000434: 3080  adds r0 #80
20000436: 0100  lsls r0 r0 #4
20000438: 6983  ldr r3 [ r0 #18 ]
2000043A: 2280  movs r2 #80
2000043C: 0052  lsls r2 r2 #1
2000043E: 4313  orrs r3 r2
20000440: 6183  str r3 [ r0 #18 ]
20000442: 3F04  subs r7 #4
20000444: 603E  str r6 [ r7 #0 ]
20000446: B430  push { r4  r5 }
20000448: 2400  movs r4 #0
2000044A: 25FF  movs r5 #FF
2000044C: 022D  lsls r5 r5 #8
2000044E: 35FF  adds r5 #FF
20000450: 012D  lsls r5 r5 #4
20000452: 350F  adds r5 #F
20000454: CF40  ldmia r7 { r6 }
20000456: 3401  adds r4 #1
20000458: 42AC  cmp r4 r5
2000045A: D1FC  bne 20000456
2000045C: BC30  pop { r4  r5 }
2000045E: 2090  movs r0 #90
20000460: 04C0  lsls r0 r0 #13
20000462: 3080  adds r0 #80
20000464: 0100  lsls r0 r0 #4
20000466: 6983  ldr r3 [ r0 #18 ]
20000468: 2280  movs r2 #80
2000046A: 0452  lsls r2 r2 #11
2000046C: 4313  orrs r3 r2
2000046E: 6183  str r3 [ r0 #18 ]
20000470: 3F04  subs r7 #4
20000472: 603E  str r6 [ r7 #0 ]
20000474: B430  push { r4  r5 }
20000476: 2400  movs r4 #0
20000478: 25FF  movs r5 #FF
2000047A: 022D  lsls r5 r5 #8
2000047C: 35FF  adds r5 #FF
2000047E: 012D  lsls r5 r5 #4
20000480: 350F  adds r5 #F
20000482: CF40  ldmia r7 { r6 }
20000484: 3401  adds r4 #1
```

```
20000486: 42AC  cmp r4 r5
20000488: D1FC  bne 20000484
2000048A: BC30  pop { r4  r5 }
2000048C: 2300  movs r3 #0
2000048E: 2B00  cmp r3 #0
20000490: D0CE  beq 20000430
20000492: BD00  pop { pc }
 ok.
```

Native Forth Blinky Written for Readability

```
: RCC_AHBENR_IOPCEN  19  lshift $40021014 ;  \ GPIOC Enable Bit
: GPIOC_MODER_MODER8 16  lshift $48000800 ;  \ GPIOC-8 is connected to blue led
: LEDON  %1  8  lshift $48000818   bis! ;    \ GPIOC_BSRR_BS8, turn on blue led
: LEDOFF %1  24  lshift $48000818  bis! ;    \ GPIOC_BSRR_BR8, turn off blue led
: DELAY $fffff 0 do loop ;                   \ delay $fffff


: blinky ( -- )
    %1  RCC_AHBENR_IOPCEN  bis!        \ enable I/O port C clock
    %01 GPIOC_MODER_MODER8  bis!       \ set GPIOC_MODER_MODER8 as OUTPUT
    begin                              \ endless loop
       LEDON
         DELAY
       LEDOFF
         DELAY
    0 until
;
```

Mecrisp-Stellaris Changelog

2.4.2

```
05.06.2013, Matthias Koch, Mecrisp-Stellaris 0.1 experimental:

  I am very happy to announce Mecrisp-Stellaris !
  This is the very first beginning, expect monsters inside.

  As this is the younger sister of Mecrisp for MSP430,
  they share their development history and most design ideas.


12.06.2013, Matthias Koch, Mecrisp-Stellaris 0.2 experimental:

  - Unicode support
  - First interrupt handler for Systick-Timer
  - Lots of small changes
  - Ledcomm ported

15.06.2013, Matthias Koch, Mecrisp-Stellaris 0.3 experimental:

  - Recognizes both LF and CR as line ending
  - Uses hardware FIFO buffer for serial communication
  - Enables GPIO hardware on startup
  - Tidied up internal register usage
  - Faster interrupt entry
  - Fetch-Modify-Store definitions are now interrupt safe
  - Mecrisp-Stellaris now generates smaller code
  - Tidied up debug messages
  - Lots of small changes

23.06.2013, Matthias Koch, Mecrisp-Stellaris 0.4 experimental:

  - Examples in Forth for Analog, Timer, PWM and Interrupts
  - Interrupt handling for Ports and Timers included
  - Tidied up source code
  - Core needs less RAM
  - Lots of small changes

01.07.2013, Matthias Koch, Mecrisp-Stellaris 0.5
experimental:

  - Faster comparisons & bit@
  - Beautiful output for .s in current base
  - Analog Interrupts for ADC0 included
  - Examples in Forth for PLL setup and 1 Msps analog
sampling

07.07.2013, Matthias Koch, Mecrisp-Stellaris 0.6
experimental:

  - New target: STM32F407 on STM32F4-Discovery board
  - Removed experimental atomic memory access from core
  - Rearranged source code and
    added English comments to simplify porting
  - First examples for STM32F4 included

29.07.2013, Matthias Koch, Mecrisp-Stellaris 0.6.1
experimental:

  - Bugfix in min and max
  - 1 Msps analog sampling example for STM

24.08.2013, Matthias Koch, Mecrisp-Stellaris 0.7
experimental:

  - New target: KL25Z128 on Freescale Freedom FRDM-KL25Z
board
  - Deep changes in core:
    - New code generator for M0
    - Improved code generator for M3

10.08.2013, Matthias Koch, Mecrisp-Stellaris 0.7.1
experimental:

  - ADC and DAC examples for KL25Z128
  - Improved code generation for M0

07.11.2013, Matthias Koch, Mecrisp-Stellaris 0.7.2:
  - A lot of testing later: "experimental" is gone !
  - Disassembler for M0 included
  - Two channel 1 Msps sampling example for LM4F120
  - Contributions:
      Random number generator for STM32F4 by Sean Stasiak
      LM4F232H5QC port for Chinese QF-LM4F232 board by □□□
(Albert)

16.11.2013, Matthias Koch, Mecrisp-Stellaris 0.8:

  - Deep changes in core: Opcodings !
    They generate faster code for all targets.

28.11.2013, Matthias Koch, Mecrisp-Stellaris 0.8.1:

  - New target: STM32F100 on STM VL Discovery board

18.12.2013, Matthias Koch, Mecrisp-Stellaris 0.9:

  - Double numbers and s31.32 fixpoint calculations
  - Sine/Cosine Cordic algorithm
  - Two channel 1 Msps oscilloscope for LM4F120

31.12.2013, Matthias Koch, Mecrisp-Stellaris 0.9.1:

  - Partial disassembler for M3/M4 included
  - Small optimisations in core

13.01.2014, Matthias Koch, Mecrisp-Stellaris 0.9.2:

  - Small optimisations in M3/M4 code generator
  - Faster interrupt trampolines for M3/M4
  - Timer interrupts for STM32F407
  - Added rdrop and 2rdrop

06.02.2014, Matthias Koch, Mecrisp-Stellaris 0.9.3:

  - Added for convenience: d/ mod f.n
  - Final testing package for upcoming stable 1.0 release

01.03.2014, Matthias Koch, Mecrisp-Stellaris 1.0:

  Finally the first stable release of Mecrisp-Stellaris !
```

```
21.03.2014, Matthias Koch, Mecrisp-Stellaris 1.0.1:

  Bugfix in third loop index "k"

26.03.2014, Matthias Koch, Mecrisp-Stellaris 1.0.2:

  - New target: TM4C1294 on Tiva Connected Launchpad

05.04.2014, Matthias Koch, Mecrisp-Stellaris 1.0.3:

  - New target: STM32F429ZIT6 on STM F429 Discovery
  - Instruction set emulator for Cortex M0 included

23.04.2014, Matthias Koch, Mecrisp-Stellaris 1.0.4:

  - Added Intel Hex style binaries for STM targets
  - Improved code generation for do and ?do
  - Fixed minor issue with arshift when opcoded
      with folding constant >= 32
  - Added bit rotations ror and rol
  - M3/M4 only:
    - Added 12bitencoding
    - Faster opcodes for inline literals,
        calculations, logic and comparisions
    - Included new opcodes into disassembler-m3

01.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.0

  Big tidyup release !
  - Improved readability of chip specific code
  - Factored out a lot of common code
  - Move now behaves well for both directions
  - Added fill and compiletoram?
  - Improved dictionary building macros
  - Fixed flaw in LM4F120 PLL example

02.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.1

  - New target: STM32F051R8 on STM F0 Discovery
  - Tidyup and factoring of common STM terminal code

03.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.2

  - New target: STM32F401RE on STM Nucleo F401RE
  - Variables and <builds does> are now 4-aligned on M3/M4,
too
  - Added buffer:
  - Hooks for terminal redirection on TM4C1294

08.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.3

  - Added sp@ sp! rp@ rp!
  - Added hook for inner quit loop
  - ?emit and hooks for terminal redirection on all targets
  - Serial terminal code calls vectorized pause for
multitasking
  - Tidyup and factoring of more common code

19.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.4

  - New target: STM32F103C8 on Shenzhen LC Technology board

21.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.5

  - Tab is now recognized as whitespace
  - Added irq-fault

27.05.2014, Matthias Koch, Mecrisp-Stellaris 1.1.6

  - Analog and PLL examples for TM4C1294
  - Enabled Prefetch Buffer in PLL examples for STM32F407
  - Added Forth source for multiline comments
  - Added Bitlog and Bitexp functions
  - Improved code generation for fetch-modify-store shortcuts
  - Testing of LM4F120 binary on TI Tiva Launchpad with
TM4C123GH6PM

03.06.2014, Matthias Koch, Mecrisp-Stellaris 1.1.7

  - Added string handling helpers: expect tib >in
  - Contribution by Mark Schweizer:
      MK20DX256VLH7 port for Teensy 3.1 board

26.06.2014, Matthias Koch, Mecrisp-Stellaris 1.1.8

  - Added Loran-C for TM4C1294
  - Contribution by John Huberts:
      Port for Nordic nRF51822
  - Contribution by Bernd Paysan:
      Ethernet drivers for TM4C1294
```

```
30.06.2014, Matthias Koch, Mecrisp-Stellaris 2.0.0

  - Big changes in string handling !
    The new core is now a lot more ANS aligned and handles addr-length strings.
  - Renamed ?key in key? and ?emit in emit?
  - Renamed old counted-string handling with c-prefix
  - Create now has default action equivalent to : create <builds does> ;
    Old behaviour is renamed into (create)
  - Added irq-terminal to LM4F120 and TM4C1294 targets
  - Fixed flaw in 16-bit Flash write emulation layer for Freescale targets
  - Changed disassemblers and examples for new string handling

  Simply renamed with same function:
     ?key          --> key?
     ?emit         --> emit?
     serial-?key   --> serial-key?
     serial-?emit  --> serial-emit?
     hook-?key     --> hook-key?
     hook-?emit    --> hook-emit?
     expect        --> cexpect
     create        --> (create)
     type          --> ctype
     s"            --> c"

  Chanced behaviour:
     string,
     find
     compare
     number
     token
     parse
     #>

     s"
     type
     create

  Freshly added:
     cell+
     cells
     align
     aligned
     spaces
     count
     create
     type
     accept
     evaluate
     s"
     current-source
     setsource
     source

02.07.2014, Matthias Koch, Mecrisp-Stellaris 2.0.0a

  - Fixed a flaw in NVIC Forth code
  - Fixed a minor flaw in ethernet driver
  - Added 1Msps analog-digital converter ethernet stream
example

08.07.2014, Matthias Koch, Mecrisp-Stellaris 2.0.0b

  - Added comments for nRF51822 in README
  - Fixed flaw in Pascal UDP ethernet terminal code

09.07.2014, Mark Schweizer, Mecrisp-Stellaris 2.0.1

  Following changes added for MK20DX256:

  - terminal.s:  Turned on UART0 RX/TX FIFO
                 Changed code for ensuring UART overrun flag
is cleared
                 Added support for hardware flow control
  - vectors.s:   Added UART0_S/E interrupt vektor to permit
the possibility
                 of interrupt driven serial (interrupts not
enabled by
                 default)
  - interrupt.s: Added "ipsr" word to push Interrupt Program
Status Register
                 on stack
                 Added initialization of UART0S / UART0E
interrupts
  - README:      Documented serial connection, flashing, etc.
(basic info)
```

```
  - Forth:       Added more examples, cleaned up code.
Seperate README
                 located in that directory.

  Matthias Koch:
  - Cleaned up core sources to save a few bytes

16.07.2014, Matthias Koch, Mecrisp-Stellaris 2.0.2

  - Added IPSR by Mark Schweizer for all targets
  - Added UNHANDLED as new default for all IRQs
  - irq-fault now catches fault handlers only
  - irq-collection catches all other interrupts
  - Factored common vector tables and interrupts
  - Added dictionarynext to core
  - Removed dump

  - New target: STM32F303VCT6 on STM F3 Discovery
  - New target: STM32L152RE   on STM Nucleo L152RE

23.08.2014, Matthias Koch, Mecrisp-Stellaris 2.0.3

  - Bugfix: Double division
  - Some small optimisations in core
  - Assembler for Cortex M0 added

18.10.2014, Matthias Koch, Mecrisp-Stellaris 2.0.4

  - Optimisations for bit@ hbit@ cbit@
  - Added rdepth rpick
  - Forth code for blocks and trace
  - Ported Ledcomm to Nucleo L152
  - Contribution by John Huberts:
      Nordic nRF51822 radio code

06.11.2014, Matthias Koch, Mecrisp-Stellaris 2.0.5

  - New experimental target: LPC1114FN28 (without Flash write
access for now)
  - Unicode block editor added

11.11.2014, Matthias Koch, Mecrisp-Stellaris 2.0.5a

  - Adapted Thumbulator emulator for LPC1114FN28 and
      added Forth code to generate custom binaries for this
```

```
target
  - Fixed a small flaw in disassemblers

04.12.2014, Matthias Koch, Mecrisp-Stellaris 2.0.6

  - Finally solved compilation into Flash for LPC1114FN28

01.01.2015, Matthias Koch, Mecrisp-Stellaris 2.0.7

  - Delete character 127 is now recognized
  - New target: STM32F030F4
  - Ethernet driver now has support for DHCP and ARP
  - Improved analog stream over ethernet
  - Added Ledcomm for LPC1114FN28
  - Added Real Time Clock for LM4F120 and TM4C1294
  - Added Inertial Sensor code for STM32F3 Discovery
  - Fixed counter rollover issues in systick timer delay
  - Contribution by Terry Porter:
      Register file generator from SVD to Forth with STM
examples

08.02.2015, Matthias Koch, Mecrisp-Stellaris 2.0.8

  - Fixed flaw in LPC1114FN28 specific flash buffer handling
    that caused freezes after calling quit
  - Multitasking example for all targets
  - PLL and lowpower terminal code for LPC1114FN28
  - Initial graphics drivers to draw pixels on STM32F429
Discovery
  - Contribution by James Bowmann:
      Forth drivers for Gameduino 2 on STM32F401RE Nucleo
      Marsaglia pseudo random number generator
  - Contribution by Terry Porter:
      Register file generator with individual bitfields

22.02.2015, Matthias Koch, Mecrisp-Stellaris 2.0.9

  - Added eint? and reset
  - Removed obsolete ?dnegate
  - Fixed some documentation flaws in glossary

13.03.2015, Matthias Koch, Mecrisp-Stellaris 2.1.0

  - New target: LPC1115FBD48
  - Contribution by Bert Lohmann:
      I2C, LCD, PWM in and PWM out examples for LPC1114FN28

04.04.2015, Matthias Koch, Mecrisp-Stellaris 2.1.1

  - New target: MSP432P401R on MSP432 Launchpad
  - Eraseflash now performs an hardware Reset on all targets

11.04.2015, Matthias Koch, Mecrisp-Stellaris 2.1.2

  - New target: STM32L053C8 on STM32L053 Discovery
  - E-Paper display example for this new board

18.04.2015, Matthias Koch, Mecrisp-Stellaris 2.1.3

  - Ethernet driver updated to new multitasker
  - Added graphics library for lines, circles, ellipses and
bitmap font
  - Contribution by Jean Jonethal:
      New target: Nucleo STM32F411
  - Contribution by Glen Worstell:
      An example for a custom prompt

27.09.2015, Matthias Koch, Mecrisp-Stellaris 2.1.4

  - New experimental target: ARM-Linux and Android
  - Faster implementation of abs function
  - Jean Jonethal fixed UART blocking on overrun for
STM32F303
  - Added early experimental bignum implementation
  - Contribution by Ulrich Hoffman:
      Examples for syscall usage in ARM Linux

15.10.2015, Matthias Koch, Mecrisp-Stellaris 2.1.5

  - Contribution by Абдрахимов Илья:
      New target: STM L152 Discovery

22.10.2015, Matthias Koch, Mecrisp-Stellaris 2.2.0 & RA 1.0
experimental

  - Merged the experimental RA branch with the mainstream
package

  Changes to the mainstream release are:
```

```
  - Replaced align, with halign
  - Removed align4, as it has been identical with align
  - Improved maintainability of the code
  - Small improvements under the hood here and there

28.10.2015, Matthias Koch, Mecrisp-Stellaris 2.1.6

  - Many new examples for Teensy 3.1 by Andreas Wagner
  - Bugfix for STM32L152 vector map and additional interrupt
vectors,
    more examples for STM L152 Discovery by Абдрахимов Илья
(Ilya Abdrahimov)

31.10.2015, Matthias Koch, Mecrisp-Stellaris RA 1.1

  - Merged news from 2.1.6 into this package
  - Improved tuck

01.11.2015, Matthias Koch, Mecrisp-Stellaris RA 1.2

  - Jean Jonethal added STM32F303-RA and fixed a bug.

05.11.2015, Matthias Koch, Mecrisp-Stellaris RA 1.3

  - Inline-Cache allows register allocation across inlined
definitions

08.11.2015, Matthias Koch, Mecrisp-Stellaris RA 1.4

  - Small improvements for negate

14.11.2015, Matthias Koch, Mecrisp-Stellaris RA 1.5

  - Smaller disassemblers
  - Continuous sampling with exponential average for Loran-C
  - RTC examples for STM L152 Discovery by Абдрахимов Илья
(Ilya Abdrahimov)
  - Contribution by Jean Jonethal:
      New target: STM L476 Discovery

28.11.2015, Matthias Koch, Mecrisp-Stellaris RA 1.5a

  - Added fast integer square root
  - More examples for STM L152 Discovery by Абдрахимов Илья
(Ilya Abdrahimov)
```

```
  - Experimental new target: XMC1100 on Infineon XMC2GO board


24.12.2015, Matthias Koch, Mecrisp-Stellaris 2.2.0 -
Christmas release !

  My Christmas present for you:
  Mecrisp-Stellaris RA with Register Allocator is ready for
mainstream release
  and should already be quite stable after a few months of
testing.

  Now you can choose for your projects if you need the
smaller classic core
  or the larger RA which includes much stronger
optimisations.

  Both should have exactly the same capabilities and
behaviours.

  Some hints for the choice:

  If you touch performance barriers with the classic core,
  if you love experiments and if you are curious,
  try the fresh Mecrisp-Stellaris RA core.

  If you have something that already works well with older
releases,
  if your project has high demands on stability,
  if space is what you wish to save,
  use the smaller and rock solid classic Mecrisp-Stellaris.

  Changes relative to Mecrisp-Stellaris 2.1.6:

  - Replaced align, with halign
  - Removed align4, as it has been identical with align

01.01.2016, Matthias Koch, Mecrisp-Stellaris 2.2.1 - 32C3
release !

  E-Paper graphics buffer and paint primitives
  New target: Silicon Labs Giant Gecko Starter Kit with
EFM32GG990F1024
  Fixed flaw in RA core when inlining empty definitions
  - Contribution by Ilya Abdrahimov: 1-Wire for STM32L152RB
  - Contribution by Jean Jonethal:
      New target: STM F746 Discovery

13.02.2016, Matthias Koch, Mecrisp-Stellaris 2.2.1a

  - LDMIA + Interrupt silicon bug found: KNOWN-ISSUES added
  - Added RA for LPC1115
  - Contribution by Ralph Sahli: Peripheral libraries for
STM32F407
  - Contribution by Terry Porter: Register definitions and
insight generator
  - Contribution by Igor OM1ZZ: Eight queens bechmark and PLL
code

20.02.2016, Matthias Koch, Mecrisp-Stellaris 2.2.2

  - More interrupts in STM32F103
  - Added STM32F103-RA
  - Contribution by Ilya Abdrahimov: Many examples for
STM32F429
  - Contribution by Ralph Sahli: Many more examples and
libraries for STM32F407

26.03.2016, Matthias Koch, Mecrisp-Stellaris 2.2.3

  - New target: Silicon Labs Happy Gecko Starter Kit with
EFM32HG322F64

28.03.2016, Matthias Koch, Mecrisp-Stellaris 2.2.4

  - No line break directly after the "Redefine..." message
anymore
  - Graphics driver for Sharp LCD on Happy Gecko
  - New target: KL46Z256 on Freescale Freedom FRDM-KL46Z
board

06.04.2016, Matthias Koch, Mecrisp-Stellaris 2.2.5

  - Added ahead and changed control structures to allow more
combinations
  - Fixed Forth register constants for PORTE on KL25Z128 and
KL46Z256
  - New target: MK64FN1M0 on Freescale Freedom FRDM-K64F
board
```

```
  - New target: LPC1114FBD48
  - New target: Nordic nRF51822 prepared for Microbit

08.05.2016, Matthias Koch, Mecrisp-Stellaris 2.2.6

  - Scripts for invoking Mecrisp-Stellaris for ARM Linux with
qemu-arm-static
  - Contribution by Bernd Paysan:
      New target: STM32F207ZG on Nucleo-F207ZG

14.05.2016, Matthias Koch, Mecrisp-Stellaris 2.2.7

  Linux target got an improving touch:

    - Added cacheflush to solve difficulties with data and
instruction cache flushes
    - Welcome message is omitted when command line arguments
are specified
    - A check for Ctrl+D and for EOF in serial-key quits
Mecrisp-Stellaris for Linux
    - arguments, reset, eraseflash and eraseflashfrom added
    - Now 1 MB for compiletoram and compiletoflash each
    - Smaller binary
    - Forth code for cornerstone added
    - Handling of command line arguments in Forth

19.06.2016, Matthias Koch, Mecrisp-Stellaris 2.2.8

  - Increased stack size for Linux target
  - Mecrisp-Stellaris-Linux now uses software division like
the M0,
      as, crazy enough, there is no integer division on large
ARMs like A9 and A15.

23.08.2016, Matthias Koch, Mecrisp-Stellaris 2.2.9

  - Contribution by Ralph Sahli:
    New target: STM32F303K8 with many examples.
                "Neopixel" driver for WS2812B LEDs and more.

25.09.2016, Matthias Koch, Mecrisp-Stellaris 2.3.0

  - With a lot of heavy testing out in the wild, now it is
time to announce
    the register allocator "RA" as stable and ready for
```

```
general usage.

  - "Dotstar" driver for APA102 LED strings connected to a LM4F120
  - Profiler written in Forth to count usage of definitions
  - Forth code for conditional compilation
  - Contribution by Ilya Abdrahimov:
      Improved 1-Wire drivers and many new examples for STM32F103 and STM32F411

02.12.2016, Matthias Koch, Mecrisp-Stellaris 2.3.1

  - A bug was found in the register allocator: Fixed abs for all RA targets.
  - Contribution by Terry Porter:
      svd2forth-v2 - A fresh register file generator from SVD to Forth

15.12.2016, Matthias Koch, Mecrisp-Stellaris 2.3.2

  - RA binary for STM32L053C8
  - Contribution by Ronny Suy:
      New experimental target: ATSAME70Q21 on Atmel SAME70 X-plained,
      just still without the possibility to compile into flash.

01.01.2017, Matthias Koch, Mecrisp-Stellaris 2.3.3 - 33C3 release !

  - Added forgetram to all targets
  - Added call trace utility and quotations
  - RA binary for MSP432P401R
  - Added autoerase macro which clears flash dictionary on the first run
  - All EFM32 Gecko and MSP432 targets are now equipped with the autoerase feature
  - Contribution by Ralph Sahli:
      New targets: STM32F303RE on STM Nucleo F303RE
                   STM32L432KC on STM Nucleo L432KC

21.01.2017, Matthias Koch, Mecrisp-Stellaris 2.3.4

  - RA binary for STM32F407
  - Contribution by Ralph Sahli:
      New target: STM32F401CD on Espruino Pico

22.02.2017, Matthias Koch, Mecrisp-Stellaris 2.3.4a

  No changes in binaries, just fixed a documentation flaw and
new Forth features:

  - Fixed glossary: c-addr length strings are used by number

  - Jean-Claude Wippler conquered USB: Drivers for STM32F103
and STM32F303 included
  - Direct digital synthesis dual sinewave function generator
for STM32F407
  - Very early experimental access to the hardware floating
point unit

22.02.2017, Matthias Koch, Mecrisp-Stellaris 2.3.5

  - Found a bug: If the "init" mechanism was used on RA
cores, compilation of the first
    definition after a Reset might generate a leading bogus
sequence of opcodes due to
    a not properly initialised internal variable.

    Workaround, if you cannot update the core: Type "quit"
before doing fresh compilations.

16.03.2017, Matthias Koch, Mecrisp-Stellaris 2.3.6

  - Graphics routines now draw Unicode characters.
    Glyphs for ISO-8859-1 and ISO-8859-15 charset coverage
included, you can add more.
  - Contribution by Werner Urech:
      New target: STM32L073RZ on STM Nucleo L073RZ

28.06.2017, Matthias Koch, Mecrisp-Stellaris 2.3.7

  - RA binary for STM32F051
  - More readable cordic code with arctangent
  - Routing tool to replace running definitions for debugging
  - Multitask debug tools
  - Contribution by Terry Porter:
      Svd2gas - converts SVD register definition files into
assembler include files
  - Contribution by Ralph Sahli:
      New target: STM32F042F6

23.07.2017, Matthias Koch, Mecrisp-Stellaris 2.3.8

  - Bugfix for all multiplications involving double numbers
on M0 targets. Only ud* was fine.
  - Added hook-find and (find) to RA cores

09.08.2017, Matthias Koch, Mecrisp-Stellaris 2.3.9

  - Bugfix for d< d> du< du>

03.09.2017, Matthias Koch, Mecrisp-Stellaris 2.3.9a

  - Contribution by Manfred Mahlow:
      Vocabularies and classes for Mecrisp-Stellaris RA
      on targets which allow separate 16 bit writes into
flash memory
      which includes all TI, Silabs and STM32F... chips.

06.01.2018, Matthias Koch, Mecrisp-Stellaris 2.3.9b - 34C3
release !

  - Analog and blinky examples for "Calliope mini" board
which runs fine with the Microbit binary.
  - New Thumbulator with partial M3 support to generate cores
with precompiled sources.
    Currently available for LM4F120, TM4C1294, STM32F051,
STM32L053, STM32F103 and STM32F303.
    Added first special autogenerated binary with USB drivers
for STM32F3 Discovery board.
  - Contribution by Jean-Claude Wippler:
    Alternative M0 core emulator for Thumbulator adapted from
Pinkysim.

28.01.2018, Matthias Koch, Mecrisp-Stellaris 2.4.0

  - New experimental target: cy8c4245axi (9600 baud bit-bang
terminal and without flash write access for now)
  - Contribution by Andrew Palm:
      New target: STM32F030K6

11.02.2018, Matthias Koch, Mecrisp-Stellaris 2.4.1

  - Bugfix for <builds does> create on M3 and M4 cores with
```

```
16 bytes at once flash buffering
  - ." c" s" are now properly flagged as compile-only
  - Core version is now printed in words listing
  - Contribution by Terry Porter: Fresh SVD2Forth utility
  - Contribution by Andrew Palm:
      - Example collection for STM32F051
      - Trig and SQRT Forth code for s31.32 fixpoint


25.03.2018, Matthias Koch, Mecrisp-Stellaris 2.4.2

  - Defined VALUE and TO in Forth
  - Added brightness measurement with a LED for STM32F051
  - Contribution by Andrew Palm:
      - Improved trig and sqrt code
      - Sunset and sunrise calculations
  - Contribution by juju2013@github:
      New target: Nuvoton NUC123
```

Mecrisp-Stellaris Known Issues

2.4.2

```

\ -------------------------------------------------------------------------------
\  Known issues
\ -------------------------------------------------------------------------------

**********************************
*  LDMIA + Interrupts in M3/M4:
**********************************

  LDMIA and some other opcodes with a register list can be interrupted and
  continued afterwards in M3 and M4 cores.

  Unfortunately, there seems to be a silicon bug in some chips / core revisions
  that causes an interrupted LDMIA opcode to fail.

  The error has already been observed in LM4F120, TM4C1294,
STM32F407
  when using the multitasking example. M0 chips and
EFM32GG990 are fine.

  It could affect other M4 chips and maybe M3, too.

  So if you get mysterious crashes when using Interrupts om
M3/M4,
  try disabling preemption capabilities with this workaround:

  1 $E000E008 !

  Drawback is that this increases interrupt latency.

  Manual snipplets:

  B1.5.10 Exceptions in LDM and STM operations

       In order to allow implementations to have the best
possible interrupt response, an interrupt can be taken
       during an LDM or STM and continued after the return
from the interrupt. The continuation state of the LDM or
       STM is held in the ICI bits in the EPSR (see The
special-purpose program status registers (xPSR) on
       page B1-8). It is IMPLEMENTATION DEFINED when
interrupts are recognized, so the use of the ICI bits is
       IMPLEMENTATION DEFINED.
       The ARMv7-M architecture supports continuation of, or
restarting from the beginning, an abandoned LDM
       or STM instruction as outlined below. Where an LDM or
STM is abandoned and restarted (ICI bits are not
       supported), the instructions should not be used with
volatile memory. To support instruction replay, the LDM,
       STM, PUSH and POP instructions are required to
restore/maintain the base register when a fault occurs (no
base
       register writeback update) during execution.

  M3: http://infocenter.arm.com/help/index.jsp?
topic=/com.arm.doc.dui0552a/CHDCBHEE.html
  M4: http://infocenter.arm.com/help/index.jsp?
topic=/com.arm.doc.dui0553a/CHDCBHEE.html
```

```
  Address     Name   Type  Required privilege  Reset value
Description
  0xE000E008  ACTLR  RW    Privileged          0x00000000
Auxiliary Control Register

  Auxiliary Control Register

  [0]    DISMCYCINT     Disables interruption of multi-cycle
instructions.
         This increases the interrupt latency of the
processor because load/store and
         multiply/divide operations complete before interrupt
stacking occurs.

  M7: http://infocenter.arm.com/help/index.jsp?
topic=/com.arm.doc.dui0646b/CHDCBHEE.html
  Does not have this bit.


******************************
* Running on Android / Linux:
******************************

  The linux port is known to work on both Linux and Android,
but on some kernels and/or CPU cores,
  sudden segfaults may happen when directly executing freshly
compiled definitions.

  Cache flushing issues due to independent caches for
instructions and data may cause the problem.
```

Mecrisp-Stellaris Readme

2.4.2

```
@
@    Mecrisp-Stellaris - A native code Forth implementation
for ARM-Cortex M microcontrollers
@    Copyright (C) 2013  Matthias Koch
@
```

```
@    This program is free software: you can redistribute it
and/or modify
@    it under the terms of the GNU General Public License as
published by
@    the Free Software Foundation, either version 3 of the
License, or
@    (at your option) any later version.
@
@    This program is distributed in the hope that it will be
useful,
@    but WITHOUT ANY WARRANTY; without even the implied
warranty of
@    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
@    GNU General Public License for more details.
@
@    You should have received a copy of the GNU General
Public License
@    along with this program.  If not, see
<http://www.gnu.org/licenses/>.
@

;-------------------------------------------------------------
------------------
This is the stable release of Mecrisp-Stellaris,
a port of Mecrisp to the ARM Cortex M architecture.

It runs out of the box on:

  - TI Stellaris/Tiva Launchpad  with LM4F120H5QR /
TM4C123GH6PM
  - TI Tiva Connected Launchpad  with TM4C1294NCPDT
  - TI MSP432 Launchpad          with MSP432P401R
  - STM L053 Discovery           with STM32L053C8T6
  - STM Nucleo L152RE            with STM32L152RE
  - STM Nucleo 401RE             with STM32F401RE
  - STM Nucleo 411RE             with STM32F411RET6
  - STM F0 Discovery             with STM32F051R8
  - STM VL Discovery             with STM32F100RB
  - Shenzhen LC Technology board with STM32F103C8T6
  - STM F3 Discovery             with STM32F303VCT6
  - STM F4 Discovery             with STM32F407VGT6
  - STM F429 Discovery           with STM32F429ZIT6
  - Freescale Freedom FRDM-KL25Z with KL25Z128VLK4
  - Freescale Freedom FRDM-KL46Z with KL46Z256VLL4
  - Freescale Freedom FRDM-K64F  with MK64FN1M0VLL12
  - Infineon XMC2GO              with XMC1100Q024F0064
  - EFM32 Giant Gecko            with EFM32GG990F1024
  - EFM32 Happy Gecko            with EFM32HG322F64

  - Breadboard friendly, DIP packaged LPC1114FN28
  -                                   LPC1114FBD48
  -                                   LPC1115FBD48
  - Very small TSSOP20 packaged       STM32F030F4

Experimental:

  - Cypress CY8CKIT-049 4200     with CY8C4245AXI

Contributions:

  - Teensy 3.1                   with MK20DX256VLH7
  - Chinese QF-LM4F232           with LM4F232H5QC
  - nRFgo Starter Kit & Microbit with nRF51822
  - STM L152 Discovery           with STM32L152RBT6
  - STM Nucleo L073RZ            with STM32L073RZ
  - STM Nucleo F207ZG            with STM32F207ZG
  - STM Nucleo F303K8            with STM32F303K8
  - STM Nucleo F303RE            with STM32F303RE
  - Espruino Pico                with STM32F401CD
  - STM Nucleo L432KC            with STM32L432KC
  - STM L476 Discovery           with STM32L476VG
  - STM F746 Discovery           with STM32F746NG
  - Atmel SAME70 X-plained       with ATSAME70Q21
  -                                   STM32F042F6
  -                                   STM32F030K6
  - Nutiny-SDK-NUC123            with NUC123

http://mecrisp.sourceforge.net/

m-atthias@users.sf.net
;-------------------------------------------------------------
------------------

Mecrisp finally got a younger sister and mastered the jump
to the ARM Cortex M instruction set.

They share most of the design ideas, so Mecrisp-Stellaris can
compile
directly into Flash, generates native code with constant
```

```
folding and
inlining of short words.

Communication is via serial 115200 baud 8N1 over
  - UART0  in LM4F120   (connected to USB-serial bridge on
Stellaris Launchpad)
  - UART0  in TM4C1294  (connected to USB-serial bridge on
Tiva Connected Launchpad)
  - USCI0  in MSP432P401R  (TX on P1.3, RX on P1.2, connected
to USB-serial bridge)
  - UART   in LPC1114FN28  (TX on P1.7, RX on P1.6)
  - UART   in LPC1114FBD48 (TX on P1.7, RX on P1.6)
  - UART   in LPC1115FBD48 (TX on P1.7, RX on P1.6)
  - USART1 in STM32F051    (TX on PA9, RX on PA10)
  - USART1 in STM32F030F4  (TX on PA9, RX on PA10)
  - USART1 in STM32F030K6  (TX on PA9, RX on PA10)
  - USART1 in STM32F100    (TX on PA9, RX on PA10)
  - USART1 in STM32F103    (TX on PA9, RX on PA10)
  - USART1 in STM32F303    (TX on PA9, RX on PA10)
  - USART1 in STM32F429    (TX on PA9, RX on PA10)
  - USART1 in STM32L053C8  (TX on PA9, RX on PA10)
  - USART2 in STM32F401    (TX on PA2, RX on PA3 connected to
USB-serial bridge on Nucleo board)
  - USART2 in STM32F407    (TX on PA2, RX on PA3)
  - USART2 in STM32F411    (TX on PA2, RX on PA3 connected to
USB-serial bridge on Nucleo board)
  - USART2 in STM32L152    (TX on PA2, RX on PA3 connected to
USB-serial bridge on Nucleo board)
  - USART2 in STM32L476    (TX on PD5, RX on PD6 connected to
USB-serial bridge on Discovery board)
  - UART0  in KL25Z128     (connected to USB-serial bridge on
Freedom board)
  - UART0  in KL46Z256     (connected to USB-serial bridge on
Freedom board)
  - UART5  in LM4F232
  - UART0  in MK20DX256 (RX1/TX1 on Teensy 3.1.  Physical
pins #2/#3 (these are also Digital IO 0/1 lines)
                        Digital IO pins 18/19 are setup for
CTS/RTS respectively
  - UART0  in NUC123 (64pin and 48pin only)

;------------------------------------------------------------
------------------
Hardware and configuration for LM4F120 and TM4C123:
;------------------------------------------------------------
```

```
------------------
  Connect your cable to the Debug-USB-Port,
  set "PWR SELECT" switch to DEBUG and
  close VDD jumper.

  Flashing is possible with lm4flash:
  https://github.com/utzig/lm4tools

  On startup, this runs with internal PIOSC at 16 MHz,
  which is specified +-3% over whole temperature range.

;------------------------------------------------------------
------------------
Hardware and configuration for TM4C1294:
;------------------------------------------------------------
------------------
  Connect your cable to the Debug-USB-Port,
  set "POWER SELECT" jumpers (JP1) to "ICDI",
  close both "MCU 3V3" (JP2) and "+3v3" (JP3) jumpers and
  set communication jumper blocks (JP4 and JP5) both to UART
mode.

  Flashing is possible with lm4flash:
  https://github.com/utzig/lm4tools

  On startup, this runs with internal PIOSC at 16 MHz,
  which is specified +-3% over whole temperature range.

;------------------------------------------------------------
------------------
Hardware and configuration for MSP432P401R:
;------------------------------------------------------------
------------------
  Connect your cable to the USB-Port,
  set "JTAG switch" to XDS position,
  close 5V, 3V3, RXD, TXD jumpers.

  Flashing is possible with DSLite, which is part of Energia
15:
  DSLite -c MSP432P401R.ccxml -f mecrisp-stellaris-
msp432p401r.bin -l 0

  On startup, this runs with DCO on 12 MHz.

;------------------------------------------------------------
```

```
------------------
Hardware and configuration for LPC1114FN28 on breadboard:
;------------------------------------------------------------
------------------
  Connect 3.3 V power across
    + Vdd (Pin 21) and Vdda (Pin 7)
    - Vss (Pin 22) and Vssa (Pin 8)
  Connect TTL serial to
    TXD on P1.7 (Pin 16) and
    RXD on P1.6 (Pin 15)
  Connect two push buttons between
    /Reset (Pin 23) and GND
    Bootloader enable on P0.1 (Pin 24) and GND

  The chip enters serial bootloader in ROM
  if Reset button is released while
  the bootloader button is hold down.

  Flashing is possible with LPC21ISP:
  http://sourceforge.net/projects/lpc21isp/

  lpc21isp -wipe mecrisp-stellaris-lpc1114fn28.hex
/dev/ttyUSB0 9600 12000

  On startup, this runs with internal RC oscillator at 12 MHz
  which is specified +-1% over whole temperature range.

;------------------------------------------------------------
------------------
Hardware and configuration for STM32L053C8:
;------------------------------------------------------------
------------------
  Connect USB cable to ST-Link USB port.
  Terminal is included into the debug section and can be used
  if you close solder bridges SB2 and SB3.

  Flashing is possible with st-flash:
  https://github.com/texane/stlink

  Flash memory is mirrored in hardware starting from address
0. Mecrisp uses
  the low address range to generate short opcodes for calling
core words, but
  for flashing the binary, you have to use the "true address"
$08000000.
```

```
  st-flash erase
  st-flash write mecrisp-stellaris-stm32l053c8.bin 0x08000000

  On startup, this runs with internal 16 MHz HSI clock.

;------------------------------------------------------------
------------------
Hardware and configuration for STM32L152:
;------------------------------------------------------------
------------------
  Connect USB cable to ST-Link USB port.
  Terminal is included into the debug section on Nucleo
boards.

  Flashing is possible with st-flash:
  https://github.com/texane/stlink

  Flash memory is mirrored in hardware starting from address
0. Mecrisp uses
  the low address range to generate short opcodes for calling
core words, but
  for flashing the binary, you have to use the "true address"
$08000000.
  st-flash erase
  st-flash write mecrisp-stellaris-stm32l152.bin 0x08000000

  On startup, this runs with internal 16 MHz HSI clock.

;------------------------------------------------------------
------------------
Hardware and configuration for STM32L476:
;------------------------------------------------------------
------------------
  Connect USB cable to ST-Link USB port.
  Terminal is included into the debug section on Discovery
boards.

  Flashing is possible with st-flash:
  https://github.com/texane/stlink
  on Windows using STLINK Tools from ST website:
  http://www.st.com/web/en/catalog/tools/PF258168#

  Flash memory is mirrored in hardware starting from address
0. Mecrisp uses
  the low address range to generate short opcodes for calling
```

```
core words, but
  for flashing the binary, you have to use the "true address"
$08000000.
  st-flash erase
  st-flash write mecrisp-stellaris-stm32l476.bin 0x08000000

  On startup, this runs with internal 48 MHz MSI clock.
Currently flash ECC is
  supported. (JJ:O)

;------------------------------------------------------------
------------------
Hardware and configuration for STM32F401 and STM32F411:
;------------------------------------------------------------
------------------
  Connect USB cable to ST-Link USB port.
  Terminal is included into the debug section on Nucleo
boards.

  Flashing is possible with st-flash:
  https://github.com/texane/stlink

  Flash memory is mirrored in hardware starting from address
0. Mecrisp uses
  the low address range to generate short opcodes for calling
core words, but
  for flashing the binary, you have to use the "true address"
$08000000.
  st-flash erase
  st-flash write mecrisp-stellaris-stm32f401.bin 0x08000000
  st-flash write mecrisp-stellaris-stm32f411.bin 0x08000000

  On startup, this runs with internal 16 MHz HSI clock.

;------------------------------------------------------------
------------------
Hardware and configuration for STM32F407:
;------------------------------------------------------------
------------------
  Connect USB cable to ST-Link USB port,
  connect your favourite serial link to PA2 (TX) and PA3
(RX),
  keep logic voltage levels in mind !

  Flashing is possible with st-flash:
```

```
  https://github.com/texane/stlink

  Flash memory is mirrored in hardware starting from address 0. Mecrisp uses
  the low address range to generate short opcodes for calling core words, but
  for flashing the binary, you have to use the "true address" $08000000.
  st-flash erase
  st-flash write mecrisp-stellaris-stm32f407.bin 0x08000000

  On startup, this runs with external 8 MHz crystal HSE clock,
  as internal oscillator is specified -8% + 4.5% over whole temp range.

;------------------------------------------------------------------------------
Hardware and configuration for STM32F429:
;------------------------------------------------------------------------------
  Similiar to STM32F407.

  Connect USB cable to ST-Link USB port,
  connect your favourite serial link to PA9 (TX) and PA10 (RX),
  keep logic voltage levels in mind !

  st-flash erase
  st-flash write mecrisp-stellaris-stm32f429.bin 0x08000000

  On startup, this runs with external 8 MHz crystal HSE clock.

;------------------------------------------------------------------------------
Hardware and configuration for STM32F030F4:
;------------------------------------------------------------------------------

  Connect everything necessary to power up and flash the chip.
  Connect your favourite serial link to PA9 (TX) and PA10 (RX),
  keep logic voltage levels in mind !
  st-flash erase
  st-flash write mecrisp-stellaris-stm32f030f4.bin 0x08000000

  On startup, this runs with internal 8 MHz HSI clock.

;------------------------------------------------------------------------------
Hardware and configuration for STM32F051:
;------------------------------------------------------------------------------
  Similiar to STM32F407.

  Connect USB cable to ST-Link USB port,
  connect your favourite serial link to PA9 (TX) and PA10 (RX),
  keep logic voltage levels in mind !

  st-flash erase
  st-flash write mecrisp-stellaris-stm32f051.bin 0x08000000

  On startup, this runs with internal 8 MHz HSI clock.

;------------------------------------------------------------------------------
Hardware and configuration for STM32F100:
;------------------------------------------------------------------------------
  Similiar to STM32F407.

  Connect USB cable to ST-Link USB port,
  connect your favourite serial link to PA9 (TX) and PA10 (RX),
  keep logic voltage levels in mind !

  st-flash erase
  st-flash write mecrisp-stellaris-stm32f100.bin 0x08000000

  On startup, this runs with internal 8 MHz HSI clock.

;------------------------------------------------------------------------------
Hardware and configuration for STM32F303:
;------------------------------------------------------------------------------
```

```
  Similiar to STM32F407.

  Connect USB cable to ST-Link USB port,
  connect your favourite serial link to PA9 (TX) and PA10
(RX),
  keep logic voltage levels in mind !

  st-flash erase
  st-flash write mecrisp-stellaris-stm32f303.bin 0x08000000

  On startup, this runs with internal 8 MHz HSI clock.

;------------------------------------------------------------
------------------
Hardware and configuration for KL25Z128, KL46Z256, MK64FN1M0:
;------------------------------------------------------------
------------------
  Connect USB cable to SDA-USB-PORT

  Flashing is via mass storage interface -
    the board will appear as Flash Disk, just copy .bin file
in.
    Maybe you have to do an OpenSDA firmware update before.

  On startup, this runs with internal FLL at 20.97 MHz.

  If you wish to try Mecrisp-Stellaris without target
hardware,
  there is a Cortex M0 instruction set emulator prepared for
use with
  Freescale Freedom binary images. Go into
kl25z128/thumbulator
  directory, type "make", copy the binary and then type
  "thumbulator mecrisp-stellaris-kl25z128.bin" to run.

;------------------------------------------------------------
------------------
Hardware and configuration for XMC1100:
;------------------------------------------------------------
------------------
  Connect USB cable for flashing and terminal.

  Flashing is possible with JLinkExe from Segger, which is
available for Linux:
  https://www.segger.com/jlink-software.html
  "Software and documentation pack", 32 Bit TGZ archive is
fine.

  JLink_Linux_V510c_i386# ./JLinkExe -device XMC1100-0064 -if
SWD -speed 4000
    erase
    loadfile mecrisp-stellaris-xmc1100.hex
    r
    g

  OpenOCD 0.9 has support for the XMC2GO, but it failed with
the firmware version
  of my board. The command line would be something like this:

  openocd -f board/xmc-2go.cfg
  openocd -f board/xmc-2go.cfg -c "program mecrisp-stellaris-
xmc1100.hex verify reset exit"

  If you have issues with broken characters in the terminal,
  try changing the UART_FDR_STEP value in terminal.s a bit.

  To circumvent a bug in the XMC2GO serial bridge, the
baudrate is set a bit
  off the nominal value. If you connect your own terminal,
make sure to change
  it to the correct value.
    See mecrisp-stellaris-source/xmc1100/terminal.s for
hints.

  On startup, this runs with internal MCLK = PCKL = 8 MHz
clock.

;------------------------------------------------------------
------------------
Hardware and configuration for EFM32GG990 and EFM32HG322:
;------------------------------------------------------------
------------------
  Connect USB cable to DBG connector for flashing and
terminal.

  To get the mass storage flashing interface and USB-serial
bridge,
  a firmware update of the on board debugger might be
necessary.
  See instructions on MBED:
```

```
  https://developer.mbed.org/teams/SiliconLabs/wiki/Silicon-
Labs-Firmware

  Flashing is done by copying the binary into the USB drive.

  Unfortunately, flashing this way doesn't not perform a
proper mass erase.
  If you cannot get it up and running this way, you need use
a proper debugger
  and erase the whole flash properly.

  On startup, this runs with internal HFCLK = HFRCO = 14 MHz
clock.

;------------------------------------------------------------
------------------
Hardware and configuration for MK20DX256VLH7 on a Teensy 3.1
board:
;------------------------------------------------------------
------------------
  ( Contributed by Mark Schweizer )

  Connect USB cable to USB port
  Open Teensy Loader Application
(https://www.pjrc.com/teensy/loader.html) on your PC
  Select the "mecrisp-stellaris-mk20dx256.hex" file in the
mk20dx256 directory
  Select the "Auto" button (this automatically loads the coad
and resets the Teensy when the programming button is pushed)
  Press the button on the Teensy 3.1 board to initite the
bootloader
  Connect a serial cable to TX1/RX1 (physical pins #2/#3
(these are also Digital IO 0/1 lines)
  Optionally connect CTS/RTS to physical pins #25/26 (these
are also Digital IO 18/18 lines)

                  R C
                  T T
                  S S

    o o o o o o o o o o o o o o
   |--------------------------o
   |                          o
USB|       Teensy 3.1         o
   |                          o
   |--------------------------o
    o o o o o o o o o o o o o o

    G R T
    N X X
    D

  On startup, this runs with internal FLL at 20.97 MHz.  See
MCG.txt for how to switch to 96MHz

;------------------------------------------------------------
------------------
Hardware and configuration for nRF51822:
;------------------------------------------------------------
------------------
  ( Contributed by John Huberts )

  I have been using the nRFgo Motherboard fitted with a
nRF51822 module
  to do the development and testing. Programming is done
using nRFStudio
  and the Segger J-LINK LITE that comes with the dev kit.

  I was also able to debug using GDB in the nRF51 Eclipse IDE

  The motherboard has a DB9 for serial port connection but
jumpers need to
  be placed between P15 (RXD, TXD) and P9 (P2.0, P2.1). If
you don't have a
  serial port on your PC then you will need a USB-serial
cable.

  The UART can be configured to use other pins on the SOC by
changing these
  lines in in terminal.s:

  .equ RX_PIN_NUMBER, 16
  .equ TX_PIN_NUMBER, 17

;------------------------------------------------------------
------------------
Hardware and configuration for Microbit:
;------------------------------------------------------------
------------------
```

```
  Flashing can be done by just copying mecrisp-stellaris-
microbit.hex
  into the Microbit which comes with a mass storage
interface.

  Communication is done over P0.24 TX and P0.25 RX, connected
  to USB-serial-bridge on debugger section.

;------------------------------------------------------------
------------------
Hardware and configuration for Nutiny-SDK-NUC123:
;------------------------------------------------------------
------------------
  Connect 3.3V power or USB cable.

  Flashing is possible with (any clone of) st-link V2, (any
clone of) black
  magic probe, or Nuvoton's ICP software (windows only).

  Here's how with openocd and st-link v2 on linux:

  Connect SWDIO to ICE_DAT, SWCLK to ICE_CLK, GND to VSS and
3.3V to VCC if not
  powered by USB. Get OpenOCD installed, chdir to nuc123
folder and adjust
  openocd.cfg there if needed. Then launch openocd, which
should say something
  like this:

    TargetName         Type       Endian TapName
State
--  ------------------ ---------- ------ ------------------ -
-----------
 0* NuMicro.cpu        hla_target little NuMicro.cpu
unknown

  keep openocd running and telnet localhost 4444, then:

  reset halt
  flash erase_sector 0 0 135
  flash write_image mecrisp-stellaris-nuc123.bin
  reset run

  Serial console pins are automatically configured:
```

```
  PB0(RX)/PB1(TX) for 64-pin package
  PC4(RX)/PC5(TX) for 48-pin package

  33-pin package don't have UART0 but only UART1 and I don't have one, so
  it will maybe run but without serial terminal.


;------------------------------------------------------------------------------

Mecrisp-Stellaris itself is sending LF for line ending,
but recognizes both LF and CR. Invoke Picocom with something like
picocom -b 115200 /dev/ttyACM0 --imap lfcrlf,crcrlf --omap delbs,crlf

For changing line ending to CR-LF or whatever else you like,
look at the beginning of mecrisp-stellaris-....s.

Assembling is done with the GCC-Arm-Embedded toolchain:
https://launchpad.net/gcc-arm-embedded/

;------------------------------------------------------------------------------

* Fetch-Modify-Store operations like +! or bic! are not atomic.

* You can contribute your favourite hardware handling routines
    to be included in upcoming releases !

* Chip specific sources have added English comments and hints included
    for simplify porting to other ARM Cortex-M chips.

* For design ideas, have a look into Mecrisp for MSP430, too !

;------------------------------------------------------------------------------
; Turnkey applications
;------------------------------------------------------------------------------
```

If you define a word with name "init" - its latest definition is called every startup ! You you can redefine init if you like to add e.g. additional hardware setup; don't forget to give your old definition of init a call if you have one.

;--
; Porting
;--

Mecrisp-Stellaris can assemble without change for M0, M3 and M4 cores.

M0 cores lack conditional execution "ite eq...", which has to be circumvented by conditional branches. stmdb psp!, {tos} is missing, which I use to push to data stack and they have no division in hardware.

There is an assembler switch "m0core" to assemble the core with replacement code for ARM Cortex-M0 chips. Depending on your flash capabilities, there are two useful switches:

  "charkommaavailable" which is self-explaining and
  "emulated16bitflashwrites" if your particular Flash controller
                             cannot write data in separate 16-Bit chunks.

Core needs about 14 kb flash at the moment, but reserves 16 kb for upcoming additions, and needs 1.5 kb RAM for stacks and buffers plus variables for interrupt handlers plus RAM dictionary space. Bare minimum for ARM to get it running will be around 16kb/2kb, but to have fun with it and leave space for real applications, go with at least of 32 kb flash and 4 kb RAM.

Look at the flash specification for your particular chip.

The compiler only depends on 2-aligned 16-Bit flash writes one time per location, you can remove c, and halign without issues from the compiler, but 8-Bit flash writes are nice to have for compatibility with standard code.

For special cases having only 4-aligned 32-bit one-time writes to Flash, there is a hflash! emulation layer.

Flash erasing is only available for the user, it doesn't need erase cycles while compiling.

Keep these criteria in mind, pick your favourite chip and add it as your contribution. It won't be too difficult, start with an UART echo written in assembly without libraries, and I will be there for your help.

You have to change memory map, hardware initialisation and interrupt vectors, write key?, key, emit?, emit for a first reaction and flash write access for final success. There is a common mimimum set of handlers across ARM Cortex CPUs, which will help you to get it running for the first time.

Unfortunately, there is a lot of bit twiddling necessary for peripherial modules, as they differ a lot across manufacturers. ARMs seem to only share core components like CPU, interrupt controler and a very basic systick timer.

I don't have the time to support a lot of different chips with examples.

TI ADCs have a sequencer FIFO buffer, ST ADCs need a DMA channel configured.

```
TI Flash controller offers 32 bit aligned writes only, but as
many as you
wish per location until all bits are zero;
ST-M3 have 16 bit aligned Flash write only,
ST-M4 have 8 bit, 16 bit and 32 bit Flash writes available,
but only once per location.

I would like to focus on core development, support porting to
other chips and
families with adaptions in core, but leave writing
initialisation code,
testing and writing Forth examples to "family maintainers"
which are deep
into their chip of choice.

;------------------------------------------------------------
------------------
; Register allocator
;------------------------------------------------------------
------------------

It should look and feel like the classic Mecrisp-Stellaris,
but with a very important difference:

It contains an analytical compiler which keeps track
of the top five stack elements and maps them to registers
whenever possible.

To dive in, you should load the disassembler and see
definitions you have just compiled.

Two examples, on M0:

: >gray ( u -- x ) dup 1 rshift xor ;  ok.
see >gray
00006B38: 0873  lsrs r3 r6 #1
00006B3A: 405E  eors r6 r3
00006B3C: 4770  bx lr
 ok.


: bitexp ( u -- u )  ok.
  ok.
  \ Returns an integer value equivalent to  ok.
  \ the exponential. For numbers > 16,  ok.
  \ bitexp(x) approx = 2^(x/8 + 1)  ok.
  ok.
  \ B(E(x)) = x for 16 <= x <= 247.  ok.
  ok.
  dup 247 u>  \ Overflow ?  ok.
  if drop $F0000000  ok.
  else  ok.
  ok.
    dup 16 u<= if 1 rshift  ok.
               else  ok.
                 dup ( u u )  ok.
                 7 and 8 or ( u b )  ok.
                 swap ( b u )  ok.
                 3 rshift 2 - lshift  ok.
               then  ok.
  ok.
  then  ok.
  ok.
  1-foldable ;  ok.
  ok.
see bitexp
00006BA2: 2EF7  cmp r6 #F7
00006BA4: B500  push { lr }
00006BA6: D902  bls 00006BAE
00006BA8: 26F0  movs r6 #F0
00006BAA: 0636  lsls r6 r6 #18
00006BAC: E00C  b 00006BC8
00006BAE: 2E10  cmp r6 #10
00006BB0: D801  bhi 00006BB6
00006BB2: 0876  lsrs r6 r6 #1
00006BB4: E008  b 00006BC8
00006BB6: 0033  lsls r3 r6 #0
00006BB8: 2007  movs r0 #7
00006BBA: 4003  ands r3 r0
00006BBC: 2008  movs r0 #8
00006BBE: 4303  orrs r3 r0
00006BC0: 08F6  lsrs r6 r6 #3
00006BC2: 3E02  subs r6 #2
00006BC4: 40B3  lsls r3 r6
00006BC6: 461E  mov r6 r3
00006BC8: BD00  pop { pc }
```

Note that it compiles bitexp without any stack movements at all.

```
;------------------------------------------------------------
------------------
; Hacking
;------------------------------------------------------------
------------------

If you wish to dive into the assembly sources, here comes a
register map:

r0: Free scratch register            ( Saved on interrupt
entry by hardware )
r1: Free scratch register            ( Saved on interrupt
entry by hardware )
r2: Free scratch register            ( Saved on interrupt
entry by hardware )
r3: Free scratch register            ( Saved on interrupt
entry by hardware )

r4: Inner loop count                 ( Needs Push and Pop
when used otherwise )
r5: Inner loop limit                 ( Needs Push and Pop
when used otherwise )

r6=TOS: Top-Of-Stack                 ( Stack design is
interrupt safe )
r7=PSP: Parameter Stack Pointer      ( Stack design is
interrupt safe )

r8:  Unused
r9:  Unused
r10: Unused
r11: Unused
r12: Unused                          ( Saved on interrupt
entry by hardware )

r13=SP: Return Stack Pointer
r14=LR: Link Register
r15=PC: Program Counter, always odd

;------------------------------------------------------------
------------------
Here comes a word list,
  with short descriptions of all currently included words:
View it with fixed-width font !
;------------------------------------------------------------
```

```
------------------

;------------------------------------------------------------
------------------
; Terminal-IO  (exactly ANS, some logical extensions)
;------------------------------------------------------------
------------------

        emit?           ( -- Flag ) Ready to send a character
?
        key?            ( -- Flag ) Checks if a key is
waiting
        key             ( -- Char ) Waits for and fetches the
pressed key
        emit            ( Char -- ) Emits a character.

        hook-emit?      ( -- a-addr ) Hooks for redirecting
        hook-key?       ( -- a-addr )   terminal IO
        hook-key        ( -- a-addr )     on the fly
        hook-emit       ( -- a-addr )

        serial-emit?    ( -- Flag )  Serial interface
        serial-key?     ( -- Flag )    terminal routines
        serial-key      ( -- Char )      as default
communications
        serial-emit     ( Char -- )

        hook-pause      ( -- a-addr ) Hook for a multitasker
        pause           ( -- )        Task switch, none for
default

;------------------------------------------------------------
------------------
; Stack Jugglers  (exactly ANS, some logical extensions)
;------------------------------------------------------------
------------------

Single-Jugglers:

        depth           ( -- +n ) Gives number of single-cell
stack items.
        nip             ( x1 x2 -- x2 )
        drop            ( x -- )
        rot             ( x1 x2 x3 -- x2 x3 x1 )
        -rot            ( x1 x2 x3 -- x3 x1 x2 )
        swap            ( x1 x2 -- x2 x1 )
        tuck            ( x1 x2 -- x2 x1 x2 )
        over            ( x1 x2 -- x1 x2 x1 )
        ?dup            ( x -- 0 | x x )
        dup             ( x -- x x )
        pick            ( ... xi+1 xi ... x1 x0 i -- ... x1
x0 xi )
                                  Picks one element from deep
below

        >r              ( x -- ) (R: -- x )
        r>              ( -- x ) (R: x -- )
        r@              ( -- x ) (R: x -- x )
        rdrop           (  --  ) (R: x -- )
        rdepth          ( -- +n ) Gives number of return
stack items.
        rpick           ( i -- xi ) R: ( ... xi ... x0 -- ...
xi ... x0 )

Double-Jugglers:        They perform the same for double
numbers.

        2nip            ( x1 x2 x3 x4 -- x3 x4 )
        2drop           ( x1 x2 -- )
        2rot            ( x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1
x2 )
        2-rot           ( x1 x2 x3 x4 x5 x6 -- x5 x6 x1 x2 x3
x4 )
        2swap           ( x1 x2 x3 x4 -- x3 x4 x1 x2 )
        2tuck           ( x1 x2 x3 x4 -- x3 x4 x1 x2 x3 x4 )
        2over           ( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 )
        2dup            ( x1 x2 -- x1 x2 x1 x2 )

        2>r             ( x1 x2 -- ) (R: -- x1 x2 )
        2r>             ( -- x1 x2 ) (R: x1 x2 -- )
        2r@             ( -- x1 x2 ) (R: x1 x2 -- x1 x2 )
        2rdrop          ( -- )       (R: x1 x2 -- )

Stack pointers:

        sp@             ( -- a-addr )  Fetch  data stack
pointer
        sp!             ( a-addr -- )  Store  data stack
pointer
        rp@             ( -- a-addr )  Fetch return stack
```

```
pointer
        rp!             ( a-addr -- )  Store return stack
pointer

;------------------------------------------------------------
------------------
; Logic  (exactly ANS, some logical extensions)
;------------------------------------------------------------
------------------

        arshift         ( x1 u -- x2 ) Arithmetric right-
shift of u bit-places
        rshift          ( x1 u -- x2 ) Logical right-shift of
u bit-places
        lshift          ( x1 u -- x2 ) Logical  left-shift of
u bit-places
        shr             ( x1 -- x2 )   Logical right-shift of
one bit-place
        shl             ( x1 -- x2 )   Logical  left-shift of
one bit-place
        ror             ( x1 -- x2 )   Logical right-rotation
of one bit-place
        rol             ( x1 -- x2 )   Logical  left-rotation
of one bit-place
        bic             ( x1 x2 -- x3 ) Bit clear, identical
to "not and"
        not             ( x1 -- x2 )   Invert all bits
        xor             ( x1 x2 -- x3 ) Bitwise Exclusive-OR
        or              ( x1 x2 -- x3 ) Bitwise OR
        and             ( x1 x2 -- x3 ) Bitwise AND
        false           ( --  0 ) False-Flag
        true            ( -- -1 ) True-Flag
        clz             ( x1 -- u ) Count leading zeros

;------------------------------------------------------------
------------------
; Calculus for single numbers  (exactly ANS, some logical
extensions)
;------------------------------------------------------------
------------------

        u/mod           ( u1 u2 -- u3 u4 ) 32/32 = 32 rem 32
Division
                                           u1 / u2 = u4
remainder u3
```

```
        /mod            ( n1 n2 -- n3 n4 ) n1 / n2 = n4 rem
n3
        mod             ( n1 n2 -- n3 ) n1 / n2 = remainder
n3
        /               ( n1 n2 -- n3 ) n1 / n2 = n3
        *               ( u1|n1 u2|n2 -- u3|n3 ) 32*32 = 32
Multiplication
        min             ( n1 n2 -- n1|n2 ) Keeps smaller of
top two items
        max             ( n1 n2 -- n1|n2 ) Keeps greater of
top two items
        umin            ( u1 u2 -- u1|u2 ) Keeps unsigned
smaller
        umax            ( u1 u2 -- u1|u2 ) Keeps unsigned
greater
        2-              ( u1|n1 -- u2|n2 ) Subtracts two,
optimized
        1-              ( u1|n1 -- u2|n2 ) Subtracts one,
optimized
        2+              ( u1|n1 -- u2|n2 ) Adds two,
optimized
        1+              ( u1|n1 -- u2|n2 ) Adds one,
optimized
        even            ( u1|n1 -- u2|n2 ) Makes even. Adds
one if uneven.
        2*              ( n1 -- n2 ) Arithmetic  left-shift
        2/              ( n1 -- n2 ) Arithmetic right-shift
        abs             ( n -- u ) Absolute value
        negate          ( n1 -- n2 ) Negate
        -               ( u1|n1 u2|n2 -- u3|n3 ) Subtraction
        +               ( u1|n1 u2|n2 -- u3|n3 ) Addition

;-------------------------------------------------------------
------------------
; Calculus involving double numbers  (exactly ANS, some
logical extensions)
;-------------------------------------------------------------
------------------

        um*             ( u1 u2 -- ud )        32*32 = 64
Multiplication
        ud*             ( ud1 ud2 -- ud3 )     64*64 = 64
Multiplication
        udm*            ( ud1 ud2 -- ud3-Low ud4-High )
64*64=128 Multiplication
```

```
        um/mod          ( ud u1 -- u2 u3 )     ud / u1 = u3
remainder u2
        ud/mod          ( ud1 ud2 -- ud3 ud4 ) 64/64 = 64 rem
64 Division
                                               ud1 / ud2 =
ud4 remainder ud3

        m*              ( n1 n2 -- d )         n1 * n2 = d
        m/mod           ( d  n1 -- n2 n3 )     d  / n1 = n3
remainder r2
        d/mod           ( d1 d2 -- d3 d4 )     d1 / d2 = d4
remainder d3
        d/              ( d1 d2 -- d3 )        d1 / d2 = d3
        */              ( n1 n2 n3 -- n4 )     n1 * n2 / n3 =
n4
        u*/             ( u1 u2 u3 -- u4 )     u1 * u2 / u3 =
u4
        */mod           ( n1 n2 n3 -- n4 n5 )  n1 * n2 / n3 =
n5 remainder n4
        u*/mod          ( u1 u2 u3 -- u4 u5 )  u1 * u2 / u3 =
u5 remainder u4

        d2*             ( d1 -- d2 ) Arithmetic  left-shift
        d2/             ( d1 -- d2 ) Arithmetic right-shift
        dshl            ( ud1 -- ud2 ) Logical left-shift,
same as d2*
        dshr            ( ud1 -- ud2 ) Logical right-shift

        dabs            ( d -- ud ) Absolute value
        dnegate         ( d1 -- d2 ) Negate
        d-              ( ud1|d1 ud2|d2 -- ud3|d3 )
Subtraction
        d+              ( ud1|d1 ud2|d2 -- ud3|d3 ) Addition
        s>d             ( n -- d ) Makes a signed single
number double length

;------------------------------------------------------------
------------------
; Comparisions  (exactly ANS, some logical extensions)
;------------------------------------------------------------
------------------

Single-Comparisions:
        u<=             ( u1 u2 -- flag )  Unsigned
```

```
comparisions
        u>=             ( u1 u2 -- flag )
        u>              ( u1 u2 -- flag )
        u<              ( u1 u2 -- flag )
        <=              ( n1 n2 -- flag )    Signed
comparisions
        >=              ( n1 n2 -- flag )
        >               ( n1 n2 -- flag )
        <               ( n1 n2 -- flag )
        0<              ( n - flag )         Negative ?
        0<>             ( x -- flag )
        0=              ( x -- flag )
        <>              ( x1 x2 -- flag )
        =               ( x1 x2 -- flag )

Double-Comparisions:            They perform the same for
double numbers.
        du>             ( ud1 ud2 -- flag )
        du<             ( ud1 ud2 -- flag )
        d>              ( d1 d2 -- flag )
        d<              ( d1 d2 -- flag )
        d0<             ( d -- flag )
        d0=             ( d -- flag )
        d<>             ( d1 d2 -- flag )
        d=              ( d1 d2 -- flag )

;-------------------------------------------------------------
------------------
; Tools (not only) for s31.32 fixed point numbers
(speciality!)
;-------------------------------------------------------------
------------------

Fixpoint numbers are stored ( n-comma n-whole ) and can be
handled
like signed double numbers.

        f/              ( df1 df2 -- df3 ) Division of two
fixpoint numbers
        f*              ( df1 df2 -- df3 ) Multiplication

        hold<           ( char -- )
                        Adds character to pictured number
output buffer
                        from behind.
        f#S             ( n-comma1 -- n-comma2 )
                        Adds 32 comma-digits to number output
        f#              ( n-comma1 -- n-comma2 )
                        Adds one comma-digit to number output
        f.              ( df -- )
                        Prints a fixpoint number with 32
fractional digits
        f.n             ( df n -- )
                        Prints a fixpoint number with n
fractional digits

        number          ( c-addr length -- 0 )
                                        -- n 1 )
                                        -- n-low n-high 2 )
                        Tries to convert a string to a
number.

;------------------------------------------------------------
------------------
; Number base  (exactly ANS)
;------------------------------------------------------------
------------------

        binary          ( -- ) Sets base to 2
        decimal         ( -- ) Sets base to 10
        hex             ( -- ) Sets base to 16
        base            ( -- a-addr ) Base variable address

;------------------------------------------------------------
------------------
; Memory access  (subtle differences to ANS, special cpu-
specific extensions)
;------------------------------------------------------------
------------------

        move            ( c-addr1 c-addr2 u -- ) Moves u
Bytes in Memory
        fill            ( c-addr u c ) Fill u Bytes of Memory
with value c

        cbit@           ( mask c-addr -- flag ) Test BIts in
byte-location
        hbit@           ( mask a-addr -- flag ) Test BIts in
halfword-location
        bit@            ( mask a-addr -- flag ) Test BIts in
```

```
word-location

        cxor!           ( mask c-addr -- ) Toggle bits in
byte-location
        hxor!           ( mask a-addr -- ) Toggle bits in
halfword-location
        xor!            ( mask a-addr -- ) Toggle bits in
word-location

        cbic!           ( mask c-addr -- ) Clear BIts in
byte-location
        hbic!           ( mask a-addr -- ) Clear BIts in
halfword-location
        bic!            ( mask a-addr -- ) Clear BIts in
word-location

        cbis!           ( mask c-addr -- ) Set BIts in byte-
location
        hbis!           ( mask a-addr -- ) Set BIts in
halfword-location
        bis!            ( mask a-addr -- ) Set BIts in word-
location

        2constant name  ( ud|d -- ) Makes a double constant.
        constant  name  ( u|n -- )  Makes a single constant.
        2variable name  ( ud|d -- ) Makes an initialized
double variable
        variable  name  ( n|n -- )  Makes an initialized
single variable
        nvariable name  ( n1*u|n n1 -- ) Makes an initialized
variable with
                                         specified size of n1
words
                                         Maximum is 15 words

        buffer: name    ( u -- ) Creates a buffer in RAM with
u bytes length

        2@              ( a-addr -- ud|d ) Fetches double
number from memory
        2!              ( ud|d a-addr -- ) Stores double
number in memory

        @               ( a-addr -- u|n ) Fetches single
number from memory
```

```
        !               ( u|n a-addr -- ) Stores single
number in memory
        +!              ( u|n a-addr -- ) Add to memory
location

        h@              ( c-addr -- char ) Fetches halfword
from memory
        h!              ( char c-addr ) Stores halfword in
memory
        h+!             ( u|n a-addr -- ) Add to halfword
memory location

        c@              ( c-addr -- char ) Fetches byte from
memory
        c!              ( char c-addr ) Stores byte in memory
        c+!             ( u|n a-addr -- ) Add to byte memory
location

;-------------------------------------------------------------
------------------
; Strings and beautiful output (exactly ANS, some logical
extensions)
;-------------------------------------------------------------
------------------

String routines:

        type            ( c-addr length -- )
                        Prints a string.

        s" Hello"       Compiles a string and
                        ( -- c-addr length )
                        gives back its address and length
when executed.

        ." Hello"       Compiles a string and
                        ( -- )
                        prints it when executed.

        ( Comment )     Ignore Comment
        \ Comment       Comment to end of line

        cr              ( -- ) Emits line feed
        bl              ( -- 32 ) ASCII code for Space
        space           ( -- ) Emits space
        spaces          ( n -- ) Emits n spaces if n is
positive

        compare         ( caddr-1 len-1 c-addr-2 len-2 --
flag )
                        Compares two strings

        accept          ( c-addr maxlength -- length ) Read
input into a string.

Counted string routines:

        ctype           ( cstr-addr -- )
                        Prints a counted string.

        c" Hello"       Compiles a counted string and
                        ( -- cstr-addr )
                        gives back its address when executed.

        cexpect         ( cstr-addr maxlength -- ) Read input
into a counted string.

        count           ( cstr-addr -- c-addr length )
                        Convert counted string into addr-
length string

        skipstring      ( cstr-addr -- a-addr )
                        Increases the pointer to the aligned
end of the string.

Pictured numerical output:

        .digit          ( u -- char ) Converts a digit to a
char
        digit           ( char -- u true |  false ) Converts a
char to a digit

        [char] *        Compiles code of following char
                        ( -- char ) when executed

        char *          ( -- char ) gives code of following
char
        hold            ( char -- ) Adds character to
pictured number
                                    output buffer from the
```

```
front.

        sign            ( n -- ) Add a minus sign to pictured
number
                                 output buffer, if n is
negative

        #S              ( ud1|d1 -- 0 0 ) Add all remaining
digits
                        from the double length number to
output buffer
        #               ( ud1|d1 -- ud2|d2 ) Add one digit
from the
                        double length number to output buffer
        #>              ( ud|d -- c-addr len )
                        Drops double-length number and
finishes
                        pictured numeric output ready for
type
        <#              ( -- ) Prepare pictured number output
buffer
        u.              ( u -- ) Print unsigned single number
        .               ( n -- ) Print single number
        ud.             ( ud -- ) Print unsigned double
number
        d.              ( d -- ) Print double number

Deep insights:

        words           ( -- ) Prints list of defined words.
        .s              ( many -- many ) Prints stack
contents, signed
        u.s             ( many -- many ) Prints stack
contents, unsigned
        h.s             ( many -- many ) Prints stack
contents, unsigned, hex
        hex.            ( u -- ) Prints 32 bit unsigned in
hex base,
                                 needs emit only.
                                 This is independent of
number subsystem.

;------------------------------------------------------------
------------------
; User input and its interpretation (exactly ANS, some
logical extensions)
;------------------------------------------------------------
------------------

        query           ( -- ) Fetches user input to input
buffer
        tib             ( -- cstr-addr ) Input buffer

        current-source  ( -- addr ) Double-Variable which
contains source
        setsource       ( c-addr len -- ) Change source
        source          ( -- c-addr len ) Current source
        >in             ( -- addr ) Variable with current
offset into source

        token           ( -- c-addr len ) Cuts one token out
of input buffer
        parse           ( char -- c-addr len )
                        Cuts anything delimited by char out
of input buffer

        evaluate        ( any addr len -- any ) Interpret
given string
        interpret       ( any -- any ) Execute, compile,
fold, optimize...
        quit            ( many -- ) (R: many -- ) Resets
Stacks
        hook-quit       ( -- a-addr ) Hook for changing the
inner quit loop

;------------------------------------------------------------
------------------
; Dictionary expansion  (exactly ANS, some logical
extensions)
;------------------------------------------------------------
------------------

        align           ( -- ) Aligns dictionary pointer
        aligned         ( c-addr -- a-addr ) Advances to next
aligned address
        cell+           ( x -- x+4 ) Add size of one cell
        cells           ( n -- 4*n ) Calculate size of n
cells

        allot           ( n -- ) Tries to advance Dictionary
```

```
Pointer by n bytes
                                 Aborts, if not enough space
available
        here            ( -- a-addr| c-addr )
                        Gives current position in Dictionary

        ,               ( u| n -- ) Appends a single number to
dictionary
        ><,             ( u| n -- ) Reverses high and low-
halfword, then
                                     appends it to dictionary
        h,              ( u| n -- ) Appends a halfword to
dictionary

        compiletoram?   ( -- ? ) Currently compiling into ram
?
        compiletoram    ( -- ) Makes ram   the target for
compiling
        compiletoflash  ( -- ) Makes flash the target for
compiling

        forgetram       ( -- ) Forget definitions in ram
without a reset

;-------------------------------------------------------------
------------------
; Dictionary expansion  (speciality!)
;-------------------------------------------------------------
------------------

        string,         ( c-addr len -- ) Inserts a string of
maximum 255 characters without runtime
        literal,        ( u| n -- ) Compiles a literal with
runtime
        inline,         ( a-addr -- ) Inlines the choosen
subroutine
        call,           ( a-addr -- ) Compiles a call to a
subroutine
        jump,           ( Hole-for-Opcode Destination )
                        Writes an unconditional Jump
                        to a-addr-Destination with the given
Bitmask as
                        Opcode into the halfword sized a-
addr-Hole
        cjump,          ( Hole-for-Opcode Destination Bitmask
)
                        Writes a conditional Jump
                        to a-addr-Destination with the given
Bitmask as
                        Opcode into the halfword sized a-
addr-Hole
        ret,            ( -- ) Compiles a ret opcode

        flashvar-here   ( -- a-addr ) Gives current RAM
management pointer
        dictionarystart ( -- a-addr ) Current entry point for
dictionary search
        dictionarynext  ( a-addr -- a-addr flag )
                        Scans dictionary chain and returns
true if end is reached.

    Available depending on chip capabilities:

        c,              ( char -- ) Appends a byte to
dictionary
        halign          ( -- ) Makes Dictionary Pointer even,
if uneven.

        movwmovt,       ( x Register -- ) Generate a
movw/movt-Sequence to
                                          get x into any
given Register. M3/M4 only
       registerliteral, ( x Register -- ) Generate shortest
possible sequence
                                          to get x into given
low Register.
                                          On M0: A movs-lsls-
adds... sequence
                                          M3/M4: movs / movs-
mvns / movw / movw-movt

       12bitencoding    ( x -- x false |  bitmask true )
                                          Can x be encoded as
12-bit immediate ?

;------------------------------------------------------------
------------------
; Flags and inventory  (speciality!)
;------------------------------------------------------------
------------------
```

```
        smudge          ( -- ) Makes current definition
visible, burns
                               collected flags to flash and
                               takes care of proper ending
        inline          ( -- ) Makes current definition
inlineable.
                               For flash, place it inside
your definition !
        immediate       ( -- ) Makes current definition
immediate.
                               For flash, place it inside
your definition !
        compileonly     ( -- ) Makes current definition
compileonly.
                               For flash, place it inside
your definition !
        setflags        ( char -- ) Sets Flags with a mask.
This isn't immediate,
                               but for flash, place it inside
your definition !
        (create) name   ( -- ) Creates and links a new
invisible dictionary
                               header that does nothing.
                               Use FIG-style <builds .. does>
!
        find            ( c-addr len -- a-addr flags )
                               Searches for a String in
Dictionary.
                               Gives back flags, which are
different to ANS !

        0-foldable      ( -- ) Current word becomes foldable
with zero constants
        1-foldable      ( -- ) Current word becomes foldable
with one constants
        2-foldable      ( -- ) Current word becomes foldable
with two constants
        3-foldable      ( -- ) Current word becomes foldable
with 3   constants
            ...
        7-foldable      ( -- ) Current word becomes foldable
with 7   constants

;------------------------------------------------------------
```

```
------------------
; Compiler essentials  (subtle differences to ANS)
;------------------------------------------------------------
------------------

        execute         ( a-addr -- ) Calls subroutine
        recurse         ( -- ) Lets the current definition
call itself
        ' name          ( -- a-addr ) Tries to find name in
dictionary
                                      gives back executable
address
        ['] name        ( -- a-addr)  Tick that compiles the
executable address
                                      of found word as
literal
        postpone name   ( -- ) Helps compiling immediate
words.
        does>           ( -- ) executes: ( -- a-addr )
                               Gives address to where you
have stored data.
        <builds         ( -- ) Makes Dictionary header and
reserves space
                               for special call.
        create name     ( -- ) Create a definition with
default action which
                               cannot be changed later. Use
<builds does> instead.
                               Equivalent to : create <builds
does> ;
        state           ( -- a-addr ) Address of state
variable
        ]               ( -- ) Switch to compile state
        [               ( -- ) Switch to execute state
        ;               ( -- ) Finishes new definition
        : name          ( -- ) Opens new definition

;------------------------------------------------------------
------------------
; Control structures (exactly ANS)
;------------------------------------------------------------
------------------
Internally, they have complicated compile-time stack effects.

Decisions:
```

```
flag if ... then
flag if ... else ... then

        then            ( -- )          This is the common
        else            ( -- )          flag if ... [else
...] then
        if              ( flag -- )     structure.
        ahead           ( -- )

Case:

n case
     m1   of ... endof
     m2   .. ... .....
   flag  ?of ... endof
    all others
  endcase

        case            ( n -- n )      Begins case
structure
        of              ( m -- )        Compares m with n,
choose this if n=m
        ?of             ( flag -- )     Flag-of, for custom
comparisions
        endof           ( -- )          End of one
possibility
        endcase         ( n -- )        Ends case structure,
discards n

Indefinite Loops:

begin ... again
begin ... flag until
begin ... flag while ... repeat

        repeat          ( -- ) Finish of a middle-flag-
checking loop.

        while           ( flag -- ) Check a flag in the
middle of a loop

        until           ( flag -- ) begin ... flag until
                                    loops until flag is true
        again           ( -- )  begin ... again
```

```
                                is an endless loop
        begin           ( -- )


Definite Loops:

limit index   do ... [one or more leave(s)] ... loop
             ?do ... [one or more leave(s)] ... loop
              do ... [one or more leave(s)] ... n +loop
             ?do ... [one or more leave(s)] ... n +loop


        k               ( -- u|n ) Gives third  loop index
        j               ( -- u|n ) Gives second loop index
        i               ( -- u|n ) Gives innermost loop index


        unloop          (R: old-limit old-index -- )
                        Drops innermost loop structure,
                        pops back old loop structures to loop
registers

        exit            ( -- ) Returns from current
definition.
                               Compiles a ret opcode.

        leave           ( -- ) (R: old-limit old-index -- )
                        Leaves current innermost loop
promptly

        +loop           ( u|n -- )
                        (R: unchanged | old-limit old-index -
- )
                        Adds number to current loop index
register
                        and checks whether to continue or not

        loop            ( -- )
                        (R: unchanged | old-limit old-index -
- )
                        Increments current loop index
register by one
                        and checks whether to continue or
not.
```

```
        ?do             ( Limit Index -- )
                        (R: unchanged | -- old-limit old-
index )
                        Begins a loop if limit and index are
not equal

        do              ( Limit Index -- )
                        (R: -- old-limit old-index )
                        Begins a loop

;------------------------------------------------------------
------------------
; Common Hardware access
;------------------------------------------------------------
------------------

        reset           ( -- ) Reset on hardware level
        dint            ( -- ) Disables Interrupts
        eint            ( -- ) Enables  Interrupts
        eint?           ( -- ) Are Interrupts enabled ?
        nop             ( -- ) No Operation. Hook for unused
handlers !

        ipsr            ( -- ipsr ) Interrupt Program Status
Register
        unhandled       ( -- ) Message for unhandled
interrupts.

        irq-systick     ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-fault       ( -- a-addr ) For all faults
        irq-collection  ( -- a-addr ) Collection of all
unhandled interrupts

;------------------------------------------------------------
------------------
; Specials for LM4F120:
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
```

```
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        cflash!         ( char c-addr -- )  Writes byte to
flash
        hflash!         ( u|n a-addr -- )   Writes halfword
to flash
        flash!          ( u|n 4-a-addr -- ) Writes single
number to flash,
                                              4 aligned !
Interrupts:

        irq-porta       ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-portb
        irq-portc
        irq-portd
        irq-porte
        irq-portf
        irq-timer0a
        irq-timer0b
        irq-timer1a
        irq-timer1b
        irq-timer2a
        irq-timer2b
        irq-adc0seq0
        irq-adc0seq1
        irq-adc0seq2
        irq-adc0seq3
        irq-terminal

  Look into vectors.s and interrupts.s to add more
interrupts.

;-------------------------------------------------------------
------------------
; Specials for TM4C1294:
;-------------------------------------------------------------
------------------
```

```
Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 16k flash
page only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        cflash!         ( char c-addr -- )  Writes byte to
flash
        hflash!         ( u|n a-addr -- )   Writes halfword
to flash
        flash!          ( u|n 4-a-addr -- ) Writes single
number to flash,
                                              4 aligned !
Interrupts:

        irq-porta       ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-portb
        irq-portc
        irq-portd
        irq-porte
        irq-portf
        irq-portg
        irq-porth
        irq-portj
        irq-portk
        irq-portl
        irq-timer0a
        irq-timer0b
        irq-timer1a
        irq-timer1b
        irq-timer2a
        irq-timer2b
        irq-timer3a
        irq-timer3b
        irq-adc0seq0
        irq-adc0seq1
        irq-adc0seq2
        irq-adc0seq3
        irq-adc1seq0
        irq-adc1seq1
        irq-adc1seq2
        irq-adc1seq3
        irq-ethernet
        irq-terminal

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for MSP432P401R:
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        cflash!         ( char c-addr -- )  Writes byte to
flash
        hflash!         ( u|n a-addr -- )   Writes halfword
to flash
        flash!          ( u|n 4-a-addr -- ) Writes single
number to flash,
                                              4 aligned !
Interrupts:

        irq-port1       ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-port2
        irq-port3
        irq-port4
        irq-port5
        irq-port6
        irq-watchdog
        irq-adc
        irq-timera0
        irq-timera1

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for LPC1114FN28, LPC1114FBD48 and LPC1115FBD48:
;------------------------------------------------------------
------------------

        initflash       ( -- ) Clear buffers
        hflash!         ( u|n a-addr -- ) Writes halfword to
flash (buffered)
        flushflash      ( -- ) Flush flash buffers
                                 needed if you comma in
values after create

        flash-khz       ( -- addr ) Variable with current
core frequency in kHz
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.
        16flash!        ( x1 x2 x3 x4 addr -- ) Flash aligned
16 Bytes at once

        irq-i2c         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-uart
        irq-adc

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for STM32F051:
;------------------------------------------------------------
------------------
```

```
Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-exti0_1
        irq-exti2_3
        irq-exti4_15
        irq-tim1_up
        irq-tim1_cc
        irq-tim2
        irq-tim3

  Look into vectors.s and interrupts.s to add more
interrupts.

;-------------------------------------------------------------
-----------------
; Specials for STM32F030F4:
;-------------------------------------------------------------
-----------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.
        hflash!         ( u|n a-addr -- )   Writes halfword
to flash
```

```
Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for IRQ-Hooks
        irq-exti0_1
        irq-exti2_3
        irq-exti4_15
        irq-tim1_up
        irq-tim1_cc
        irq-tim3

  Look into vectors.s and interrupts.s to add more interrupts.

;------------------------------------------------------------------------------
; Specials for STM32L053C8:
;------------------------------------------------------------------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram. Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this address.
                                      Clears Ram. Restarts Forth.

        flashpageerase  ( a-addr -- ) Erase one 128 byte flash page only. Take care:
                                      No Reset, no dictionary reinitialisation.

        flash!          ( u|n a-addr -- )   Writes word to flash

Interrupts:

        irq-rtc         ( -- a-addr ) Memory locations for IRQ-Hooks
        irq-exti0_1
        irq-exti2_3
        irq-exti4_15
        irq-touch
        irq-dma1
        irq-dma2_3
        irq-dma4_7
        irq-adc
        irq-lptim1
        irq-tim2
        irq-dac
        irq-tim21
        irq-tim22
        irq-i2c1
        irq-i2c2
        irq-spi1
        irq-spi2
        irq-usart1
        irq-usart2
        irq-rng
        irq-lcd
        irq-usb

  Look into vectors.s and interrupts.s to add more interrupts.

;------------------------------------------------------------------------------
; Specials for STM32L152:
;------------------------------------------------------------------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram. Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this address.
                                      Clears Ram. Restarts Forth.

        flashpageerase  ( a-addr -- ) Erase one 256 byte flash page only. Take care:
                                      No Reset, no dictionary reinitialisation.

        flash!          ( u| n a-addr -- )   Writes word to flash
```

```
Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for IRQ-Hooks
        irq-dac
        irq-exti4
        irq-exti3
        irq-exti2
        irq-exti1
        irq-exti0
        irq-tim2
        irq-tim3
        irq-tim4

  Look into vectors.s and interrupts.s to add more interrupts.

;------------------------------------------------------------------------------
; Specials for STM32L476:
;------------------------------------------------------------------------------

Flash:
        initflash       ( -- ) Clear buffers
        hflash!         ( u|n a-addr -- ) Writes halfword to flash (buffered)
        flushflash      ( -- ) Flush flash buffers
                               needed if you comma in values after create
        eraseflash      ( -- ) Erases everything. Clears Ram. Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this address.
                                      Clears Ram. Restarts Forth.

        flashpageerase  ( a-addr -- ) Erase one 2048 byte flash page only. Take care:
                                      No Reset, no dictionary reinitialisation.

        16flash!        ( x1 x2 x3 x4 addr -- ) Flash 4 words aligned at 8 byte boundary at once
```

```
Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-exti4
        irq-exti3
        irq-exti2
        irq-exti1
        irq-exti0
        irq-tim2
        irq-tim3
        irq-tim4

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for STM32F100:
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-exti4
        irq-exti3
        irq-exti2
        irq-exti1
        irq-exti0

  Look into vectors.s and interrupts.s to add more
interrupts.

;-------------------------------------------------------------
------------------
; Specials for STM32F103:
;-------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-exti4
        irq-exti3
        irq-exti2
        irq-exti1
        irq-exti0

        irq-rtc
        irq-exti5
        irq-tim1brk
        irq-tim1up
        irq-tim1trg
        irq-tim1cc
        irq-tim2
        irq-tim3
        irq-tim4
        irq-i2c1ev
        irq-i2c1er
        irq-i2c2ev
        irq-i2c2er
        irq-spi1
        irq-spi2
        irq-usart1
        irq-usart2
        irq-usart3
        irq-exti10
        irq-rtcalarm
        irq-usbwkup
        irq-tim5
        irq-spi3
        irq-uart4
        irq-uart5
        irq-tim6
        irq-tim7
        irq-usbfs

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for STM32F303:
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 2k flash page
only. Take care:
```

```
                                      No Reset, no dictionary
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-exti4
        irq-exti3
        irq-exti2
        irq-exti1
        irq-exti0
        irq-tim2
        irq-tim3
        irq-tim4

  Look into vectors.s and interrupts.s to add more
interrupts.

;-------------------------------------------------------------
------------------
; Specials for STM32F401, STM32F411, STM32F407 and STM32F429:
;-------------------------------------------------------------
------------------

Flash:
        eraseflash       ( -- ) Erases everything. Clears
Ram. Restarts Forth.
        eraseflashsector ( u -- ) Erases one sector of Flash

        cflash!         ( char c-addr -- )  Writes byte to
flash
        hflash!         ( u|n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-exti4
        irq-exti3
        irq-exti2
        irq-exti1
        irq-exti0
        irq-tim2
        irq-tim3
        irq-tim4

  Look into vectors.s and interrupts.s to add more
interrupts.

;-------------------------------------------------------------
------------------
; Specials for KL25Z128 and KL46Z256:
;-------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                              Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                              No Reset, no dictionary
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash - emulated
        flash!          ( u|n 4-a-addr -- ) Writes single
number to flash,
                                      4 aligned !

Interrupts:

        irq-adc         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-dac
        irq-cmp
        irq-porta
        irq-portd

  Look into vectors.s and interrupts.s to add more
```

```
interrupts.

;------------------------------------------------------------
------------------
; Specials for MK64FN1M0:
;------------------------------------------------------------
------------------

Flash:
        initflash       ( -- ) Clear buffers
        hflash!         ( u|n a-addr -- ) Writes halfword to
flash (buffered)
        flushflash      ( -- ) Flush flash buffers
                               needed if you comma in values
after create
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 4 kb flash
page only. Take care:
                                      No Reset, no dictionary
reinitialisation.

        16flash!        ( x1 x2 x3 x4 addr -- ) Flash 4 words
aligned at 8 byte boundary at once

Interrupts:

        irq-adc0         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-adc1
        irq-dac0
        irq-dac1
        irq-cmp0
        irq-cmp1
        irq-cmp2
        irq-porta
        irq-portb
        irq-portc
        irq-portd
        irq-porte

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for XMC1100:
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 256 byte
flash page only.
                                      No Reset, no dictionary
reinitialisation.

        initflash       ( -- ) Clear buffers
        hflash!         ( u|n a-addr -- ) Writes halfword to
flash (buffered)
        flushflash      ( -- ) Flush flash buffers
                                 needed if you comma in
values after create

        16flash!        ( x1 x2 x3 x4 addr -- ) Flash aligned
16 Bytes at once

Interrupts:

        irq-scu0        ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-scu1
        irq-eru0
        irq-eru1
        irq-eru2
        irq-eru3
        irq-usi0
        irq-usi1
        irq-usi2
        irq-usi3
        irq-usi4
        irq-usi5
        irq-adc0
        irq-adc1
        irq-ccu0
        irq-ccu1
        irq-ccu2
        irq-ccu3

;-------------------------------------------------------------
------------------
; Specials for EFM32GG990:
;-------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 4 kb flash
page only.
                                      No Reset, no dictionary
reinitialisation.

        hflash!         ( u| n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-dma         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-gpioeven
        irq-timer0
        irq-usart0rx
        irq-usart0tx
        irq-acmp0
        irq-adc0
        irq-dac0
        irq-i2c0
        irq-gpioodd
        irq-timer1
        irq-timer2
        irq-usart1rx
        irq-usart1tx
        irq-usart2rx
        irq-usart2tx
        irq-uart0rx
        irq-uart0tx
        irq-leuart0
        irq-leuart1
        irq-letimer0
        irq-pcnt0
        irq-pcnt1
        irq-pcnt2
        irq-rtc
        irq-cmu
        irq-vcmp
        irq-lcd
        irq-msc
        irq-aes

;-------------------------------------------------------------
------------------
; Specials for EFM32HG322:
;-------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                       Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1 kb flash
page only.
                                       No Reset, no dictionary
reinitialisation.

        hflash!         ( u| n a-addr -- )   Writes halfword
```

```
to flash

Interrupts:

        irq-dma         ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-gpioeven
        irq-timer0
        irq-acmp0
        irq-adc0
        irq-i2c0
        irq-gpioodd
        irq-timer1
        irq-usart1rx
        irq-usart1tx
        irq-leuart0
        irq-pcnt0
        irq-rtc
        irq-cmu
        irq-vcmp
        irq-msc
        irq-aes
        irq-usart0rx
        irq-usart0tx
        irq-usb
        irq-timer2

;------------------------------------------------------------
------------------
; Specials for MK20DX256:
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
                                      No Reset, no dictionary
```

```
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash - emulated
        flash!          ( u|n 4-a-addr -- ) Writes single
number to flash,
                                              4 aligned !

Interrupts:

        irq-adc0        ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-adc1
        irq-dac
        irq-cmp0
        irq-cmp1
        irq-cmp2
        irq-porta
        irq-portb
        irq-portc
        irq-portd
        irq-porte
        irq-uart0S
        irq-uart0E

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for nRF51822 and Microbit
;------------------------------------------------------------
------------------

Flash:
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        flashpageerase  ( a-addr -- ) Erase one 1k flash page
only. Take care:
```

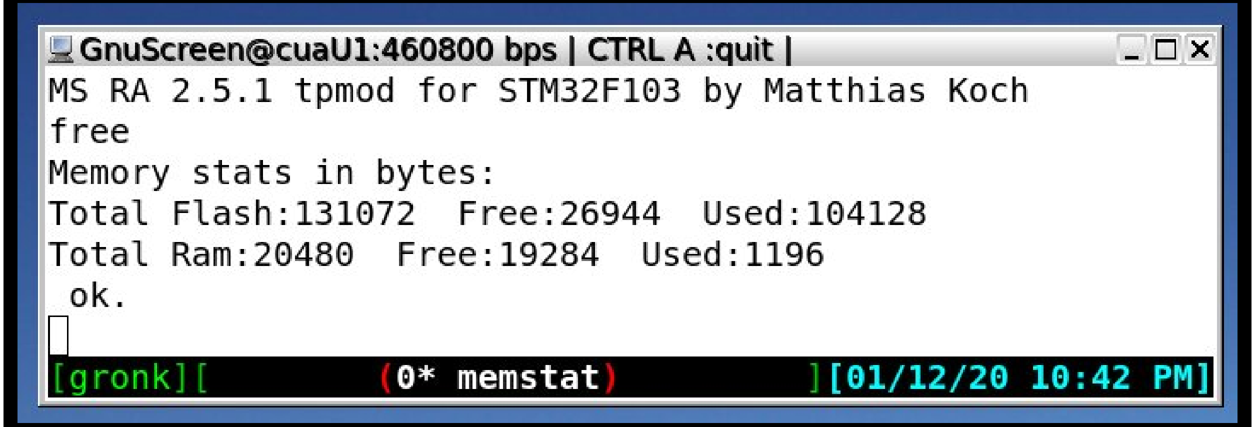
```
                                      No Reset, no dictionary
reinitialisation.

        hflash!         ( u|n a-addr -- )   Writes halfword
to flash

Interrupts:

        irq-power       ( -- a-addr ) Memory locations for
IRQ-Hooks
        irq-radio
        irq-uart
        irq-spi0
        irq-spi1
        irq-gpiote
        irq-adc
        irq-tim0
        irq-tim1
        irq-tim2
        irq-rtc0
        irq-temp
        irq-rng
        irq-ecb
        irq-ccm_aar
        irq-wdt
        irq-rtc1
        irq-qdec
        irq-lpcomp
        irq-swi0
        irq-swi1
        irq-swi2
        irq-swi3
        irq-swi4
        irq-swi5

  Look into vectors.s and interrupts.s to add more
interrupts.

;------------------------------------------------------------
------------------
; Specials for Linux
;------------------------------------------------------------
------------------

"Flash":
        eraseflash      ( -- ) Erases everything. Clears Ram.
Restarts Forth.

        eraseflashfrom  ( a-addr -- ) Starts erasing at this
address.
                                      Clears Ram. Restarts
Forth.

        cflash!         ( char c-addr -- )  Writes byte to
"flash"
        hflash!         ( u|n a-addr -- )   Writes halfword
to "flash"

System:

        cacheflush      ( -- ) Flushes data and instruction
cache
        syscall         ( r0 r1 r2 r3 r4 r5 r6 Syscall# -- r0
) Perform Syscall
        arguments       ( -- a-addr ) Gives the initial stack
pointer which
                                    contains command line
arguments.
        bye             ( -- ) Leave Mecrisp-Stellaris


Matthias Koch, Summer 2013, updated Spring 2016
```

Memstat

Memstat is a Mecrisp Forth development aid to monitor both MCU Flash and Ram memory usage.

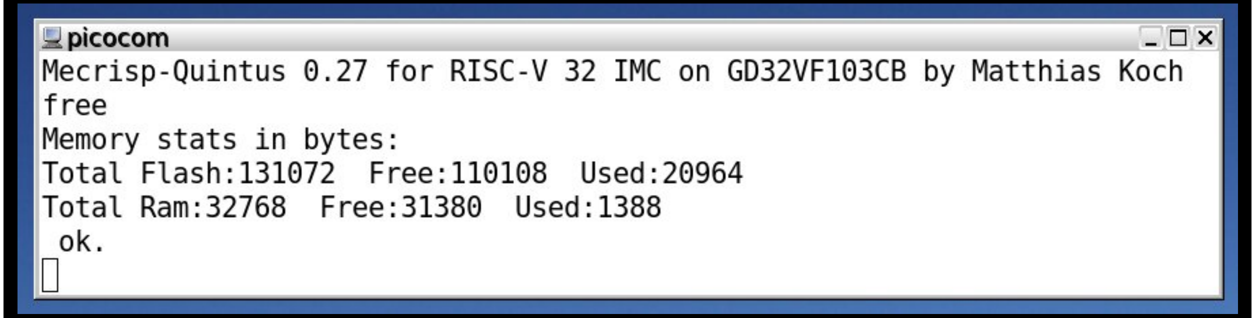
Details are in the source below.

STM32F103C8



```
MS RA 2.5.1 tpmod for STM32F103 by Matthias Koch
free
Memory stats in bytes:
Total Flash:131072  Free:26944  Used:104128
Total Ram:20480  Free:19284  Used:1196
 ok.

[gronk][          (0* memstat)            ][01/12/20 10:42 PM]
```

GD32VF103



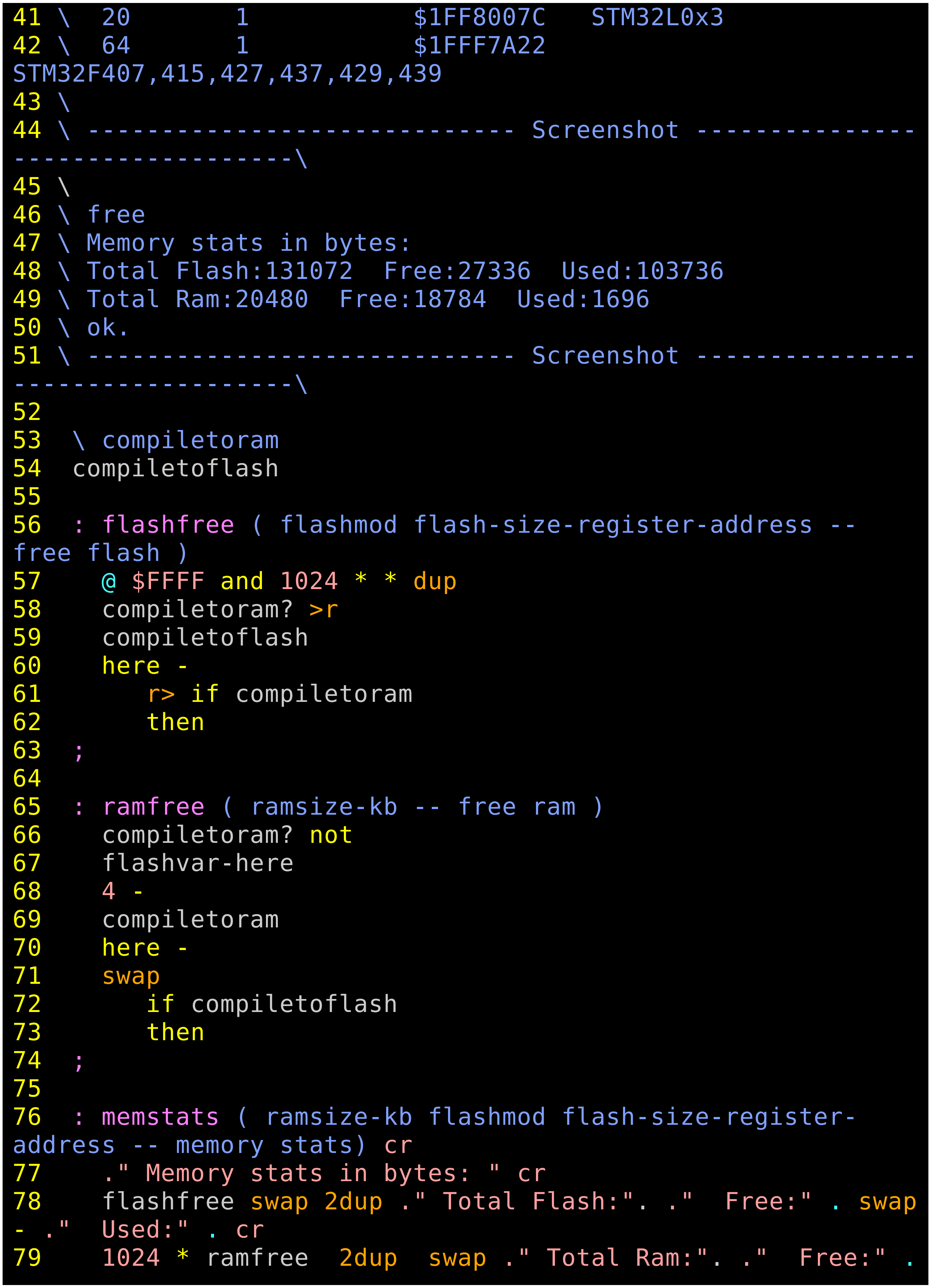
```
Mecrisp-Quintus 0.27 for RISC-V 32 IMC on GD32VF103CB by Matthias Koch
free
Memory stats in bytes:
Total Flash:131072  Free:110108  Used:20964
Total Ram:32768  Free:31380  Used:1388
 ok.
```

Code

Download memstat.fs

```
\ Program Name: memstat.fs
\ Date: Sun 12 Jan 2020 18:55:01 AEDT
\ Copyright 2020 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV2
\ For Mecrisp-Stellaris and Mecrisp-Quintus by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Standalone: no preloaded support files required
\
\ This Program : Displays memory statistics
```

```
\
\ ---------------------------------------------------------------------------\
\     Usage:    " ramsize-kb flashmod flash-size-register-address memstats "
\
\     'ramsize-kb' = Get the MCU ram size from the datasheet as it's not
\     available from the mcu directly like the flash size.
\
\     'flashmod' = flash size modifier
\        '1' = normal use, flash exactly as reported
\
\        '2' = doubles the reported flash size. Only for a STM32F103C8 which
\        reports 64kB of Flash but actually has DOUBLE (128kB) that size.
\        OR if you suspect your chip MAY have double the flash advertized. Note:
\        will crash this program with a Exception if it doesn't!
\
\     'flash-size-register-address' = "Flash size data register address". Check
\     your STM reference manual "Device electronic signature" section for the
\     correct address.
\
\     'memstats' = this program
\
\ --------------------------- MCU Type Examples -----------------------------\
\
\ ram-size  flashmod    ram-size    MCU Type
\ -kb                   -register
\                       -address
\ --------  ---------   ---------   --------
\  8        1           $1FFFF7CC   STM32F0
\  20       1           $1FFFF7E0   STM32F10
\  20       2           $1FFFF7E0   STM32F103C8 (2x indicated Flash)
\  20       1           $1FFFF7E0   CKS32F103C8T6 (Chinese STM32F103CB clone)
\  32       1           $1FFFF7E0   GD32VF103 RISC-V32, Mecrisp-Quintus
```



```
\  20       1          $1FF8007C   STM32L0x3
\  64       1          $1FFF7A22   STM32F407,415,427,437,429,439
\
\ ----------------------------- Screenshot ----------------------------------\
\
\ free
\ Memory stats in bytes:
\ Total Flash:131072  Free:27336  Used:103736
\ Total Ram:20480  Free:18784  Used:1696
\ ok.
\ ----------------------------- Screenshot ----------------------------------\

 \ compiletoram
 compiletoflash

 : flashfree ( flashmod flash-size-register-address -- free flash )
   @ $FFFF and 1024 * * dup
   compiletoram? >r
   compiletoflash
   here -
      r> if compiletoram
      then
 ;

 : ramfree ( ramsize-kb -- free ram )
   compiletoram? not
   flashvar-here
   4 -
   compiletoram
   here -
   swap
      if compiletoflash
      then
 ;

 : memstats ( ramsize-kb flashmod flash-size-register-address -- memory stats) cr
   ." Memory stats in bytes: " cr
   flashfree swap 2dup ." Total Flash:". ."  Free:" . swap - ."  Used:" . cr
   1024 * ramfree  2dup  swap ." Total Ram:". ."  Free:" .
```

```
- ."  Used:" . cr
;

: free ( -- ) 32 1 $1FFFF7E0 memstats ;  \ GD32VF103
\ : free ( -- ) 20 2 $1FFFF7E0 memstats ;  \ STM32F103C8

\ free

compiletoram
```

Warning

Ram and Flash values reported by “free” are totally seperate from values the Mecrisp-Stellaris Kernel uses. These values are preset and used when the Kernel is compiled. To configure the Kernel memory values for your MCU see Changing Kernel Flash and Ram values. Using these preset Kernel memory values, Mecrisp-Stellaris will print warnings when Flash or Ram reserves are exhausted.

Mcu Specific Special Words

LM4F120

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
cflash!	(char c-addr - -)	Writes byte to flash
hflash!	(u|n a-addr - -)	Writes halfword to flash
flash!	(u|n 4-a-addr - -)	Writes single number to flash, 4 aligned !

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-porta	(- - a-addr)	Memory locations for IRQ-Hooks
irq-portb		
irq-portc		
irq-portd		
irq-porte		
irq-portf		
irq-timer0a		
irq-timer0b		
irq-timer1a		
irq-timer1b		
irq-timer2a		

irq-timer2b		
irq-adc0seq0		
irq-adc0seq1		
irq-adc0seq2		
irq-adc0seq3		
irq-terminal		

TM4C1294

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 16k flash page only. Take care: No Reset, no dictionary reinitialisation.
cflash!	(char c-	Writes byte to flash

	addr - -)	
hflash!	(u|n a-addr - -)	Writes halfword to flash
flash!	(u|n 4-a-addr - -)	Writes single number to flash, 4 aligned !

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-porta	(- - a-addr)	Memory locations for IRQ-Hooks
irq-portb		
irq-portc		
irq-portd		
irq-porte		
irq-portf		
irq-portg		
irq-porth		

irq-portj
irq-portk
irq-portl
irq-timer0a
irq-timer0b
irq-timer1a
irq-timer1b
irq-timer2a
irq-timer2b
irq-timer3a
irq-timer3b
irq-adc0seq0

irq-adc0seq1
irq-adc0seq2
irq-adc0seq3
irq-adc1seq0
irq-adc1seq1
irq-adc1seq2
irq-adc1seq3
irq-ethernet
irq-terminal

MSP432P401R

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address.

		Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
cflash!	(char c-addr - -)	Writes byte to flash
hflash!	(u|n a-addr - -)	Writes halfword to flash
flash!	(u|n 4-a-addr - -)	Writes single number to flash, 4 aligned !

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-port1	(- - a-addr)	Memory locations for IRQ-Hooks
irq-port2		
irq-port3		
irq-port4		
irq-port5		

irq-port6
irq-watchdo
irq-adc
irq-timera0
irq-timera1

LPC1114FN28

LPC1115FBD48

Look into vectors.s and interrupts.s to add more interrupts.

initflash	(- -)	Clear buffers
hflash!	(u|n a-addr - -)	Writes halfword to flash (buffered)
flushflash	(- -)	Flush flash buffers; needed if you comma in values after create
flash-khz	(- - addr)	Variable with current core frequency in kHz

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
16flash!	(x1 x2 x3 x4 addr –)	Flash aligned 16 Bytes at once
irq-i2c irq-uart irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks

STM32F051

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.

hflash!	(u|n a-addr --)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(-- a-addr)	Memory locations for IRQ-Hooks
irq-exti0_1		
irq-exti2_3		
irq-exti4_15		
irq-tim1_up		
irq-tim1_cc		
irq-tim2		
irq-tim3		

STM32F030F4

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
hflash!	(u|n a-addr - -)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-exti0_1		
irq-exti2_3		
irq-exti4_15		
irq-tim1_up		
irq-tim1_cc		
irq-tim3		

STM32L053C8

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 128 byte flash page only. Take care: No Reset, no dictionary reinitialisation.
flash!	(u|n a-addr - -)	Writes word to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-rtc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-exti0_1		
irq-exti2_3		
irq-exti4_15		
irq-touch		

irq-dma1

irq-dma2_3

irq-dma4_7

irq-adc

irq-lptim1

irq-tim2

irq-dac

irq-tim21

irq-tim22

irq-i2c1

irq-i2c2

irq-spi1

irq-spi2

irq-usart1
irq-usart2
irq-rng
irq-lcd
irq-usb

STM32L152

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 256 byte flash page only. Take care: No Reset, no dictionary reinitialisation.
flash!	(u|n a-addr - -)	Writes word to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-dac		
irq-exti4		
irq-exti3		
irq-exti2		
irq-exti1		
irq-exti0		
irq-tim2		
irq-tim3		
irq-tim4		

STM32L476

Flash

initflash	(- -)	Clear buffers |
hflash!	(u|n a-addr - -)	Writes halfword to flash
flushflash	(- -)	Flush flash buffers needed if you comma in values after create
eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 2048 byte flash page only. Take care: No Reset, no dictionary reinitialisation.
16flash!	(x1 x2 x3 x4 addr - -)	Flash 4 words aligned at 8 byte boundary at once

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-exti4		
irq-exti3		
irq-exti2		
irq-exti1		
irq-exti0		
irq-tim2		
irq-tim3		
irq-tim4		

STM32F100

STM32F103

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.

eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr - -)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-exti4		
irq-exti3		
irq-exti2		
irq-exti1		
irq-exti0		

STM32F303

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 2k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr - -)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-exti4		
irq-exti3		
irq-exti2		

irq-exti1
irq-exti0
irq-tim2
irq-tim3
irq-tim4

STM32F401

STM32F407

STM32F411

STM32F429

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashsector	(u - -)	Erases one sector of Flash
cflash!	(char c-	Writes byte to flash

	addr - -)	
hflash!	(u|n a-addr - -)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-exti4		
irq-exti3		
irq-exti2		
irq-exti1		
irq-exti0		
irq-tim2		
irq-tim3		
irq-tim4		

KL25Z128

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr - -)	Writes halfword to flash - emulated
flash!	(u|n 4-a-addr - -)	Writes single number to flash, 4 aligned !

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc	(- - a-addr)	Memory locations for IRQ-Hooks
irq-dac		

irq-cmp		
irq-porta		
irq-portd		

XMC1100

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 256 byte flash page only. No Reset, no dictionary reinitialisation.
initflash	(- -)	Clear buffers
hflash!	(u|n a-addr - -)	Writes halfword to flash
flushflash	(- -)	Flush flash buffers needed if you comma in values after

		create
16flash!	(x1 x2 x3 x4 addr - -)	Flash aligned 16 Bytes at once

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-scu0	(- - a-addr)	Memory locations for IRQ-Hooks
irq-scu1		
irq-eru0		
irq-eru1		
irq-eru2		
irq-eru3		
irq-usi0		
irq-usi1		
irq-usi2		

irq-usi3		
irq-usi4		
irq-usi5		
irq-adc0		
irq-adc1		
irq-ccu0		
irq-ccu1		
irq-ccu2		
irq-ccu3		

EFM32GG990

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address.

		Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 4k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr - -)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-dma	(- - a-addr)	Memory locations for IRQ-Hooks
irq-gpioeven		
irq-timer0		
irq-usart0rx		
irq-usart0tx		
irq-acmp0		
irq-adc0		

irq-dac0
irq-i2c0
irq-gpioodd
irq-timer1
irq-timer2
irq-usart1rx
irq-usart1tx
irq-usart2rx
irq-usart2tx
irq-uart0rx
irq-uart0tx
irq-leuart0
irq-leuart1

irq-letimer0
irq-pcnt0
irq-pcnt1
irq-pcnt2
irq-rtc
irq-cmu
irq-vcmp
irq-lcd
irq-msc
irq-aes

MK20DX256

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.

eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.
flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr - -)	Writes halfword to flash - emulated
flash!	(u|n 4-a-addr - -)	Writes single number to flash, 4 aligned !

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

irq-adc0	(- - a-addr)	Memory locations for IRQ-Hooks
irq-adc1		
irq-dac		
irq-cmp0		
irq-cmp1		

irq-cmp2		
irq-porta		
irq-portb		
irq-portc		
irq-portd		
irq-porte		
irq-uart0S		
irq-uart0E		

nRF51822

Flash

eraseflash	(- -)	Erases everything. Clears Ram. Restarts Forth.
eraseflashfrom	(a-addr - -)	Starts erasing at this address. Clears Ram. Restarts Forth.

flashpageerase	(a-addr - -)	Erase one 1k flash page only. Take care: No Reset, no dictionary reinitialisation.
hflash!	(u|n a-addr - -)	Writes halfword to flash

Interrupts

Look into vectors.s and interrupts.s to add more interrupts.

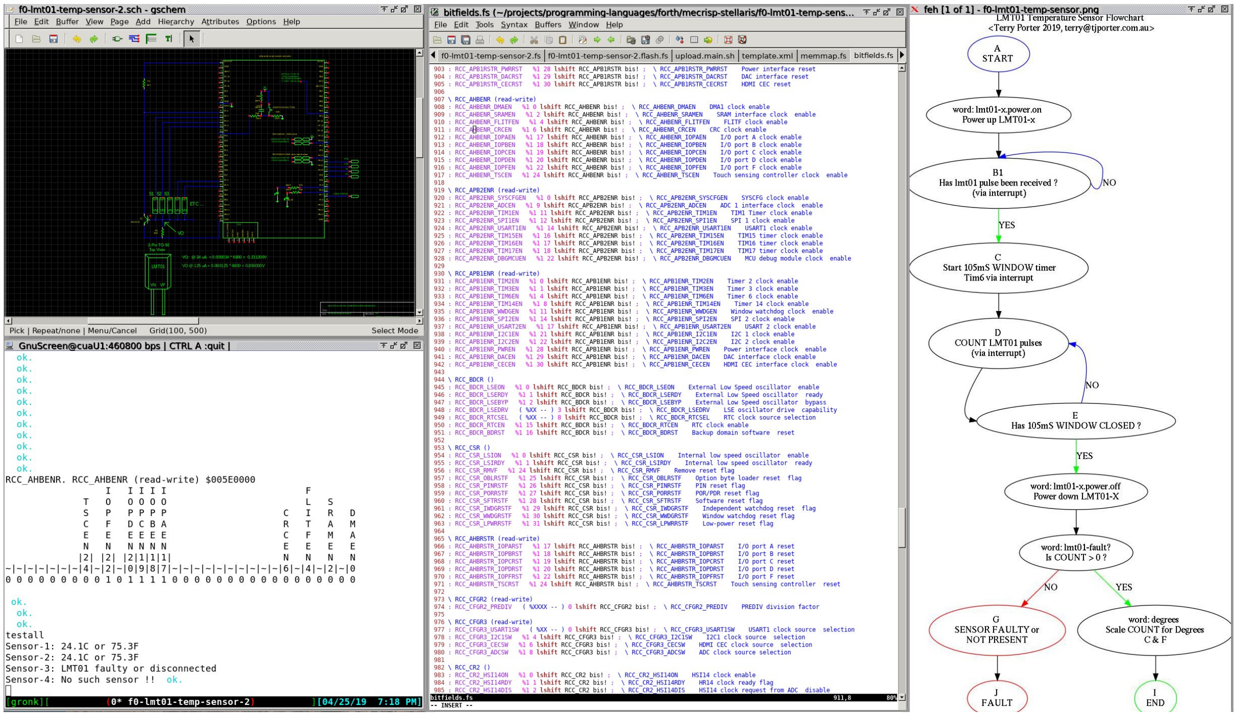
irq-power	(- - a-addr)	Memory locations for IRQ-Hooks
irq-radio		
irq-uart		
irq-spi0		
irq-spi1		
irq-gpiote		
irq-adc		
irq-tim0		

irq-tim1
irq-tim2
irq-rtc0
irq-temp
irq-rng
irq-ecb
irq-ccm_aar
irq-wdt
irq-rtc1
irq-qdec
irq-lpcomp
irq-swi0
irq-swi1

irq-swi2	
irq-swi3	
irq-swi4	
irq-swi5	

Miscellanious

A Modern Forth IDE


f0-lmt01-temp-sensor-2.sch - gschem
File Edit Buffer View Page Add Hierarchy Attributes Options Help
2-Pin TO-92
Top View
LMT01
VN VP
VO: @ 34 uA = 0.000034 * 6800 = 0.231200V
VO @ 125 uA = 0.000125 * 6800 = 0.850000V
Pick | Repeat/none | Menu/Cancel Grid(100, 500) Select Mode
GnuScreen@cuaU1:460800 bps | CTRL A :quit |

```
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
RCC_AHBENR. RCC_AHBENR (read-write) $005E0000
                  I   I I I I                         F
              T   O   O O O O                         L   S
              S   P   P P P P                     C   I   R   D
              C   F   D C B A                     R   T   A   M
              E   E   E E E E                     C   F   M   A
              N   N   N N N N                     E   E   E   E
             |2| |2| |2|1|1|1|                    N   N   N   N
~|~|~|~|~|~|~|4|~|2|~|0|9|8|7|~|~|~|~|~|~|~|~|~|~|6|~|4|~|2|~|0
0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
  ok.
  ok.
testall
Sensor-1: 24.1C or 75.3F
Sensor-2: 24.1C or 75.3F
Sensor-3: LMT01 faulty or disconnected
Sensor-4: No such sensor !!  ok.

[gronk][       (0* f0-lmt01-temp-sensor-2)        ][04/25/19  7:18 PM]
```

bitfields.fs (~/projects/programming-languages/forth/mecrisp-stellaris/f0-lmt01-temp-sens...
File Edit Tools Syntax Buffers Window Help
f0-lmt01-temp-sensor-2.fs
f0-lmt01-temp-sensor-2.flash.fs
upload.main.sh
template.xml
memmap.fs
bitfields.fs

```
: RCC_APB1RSTR_PWRRST   %1 28 lshift RCC_APB1RSTR bis! ;  \ RCC_APB1RSTR_PWRRST    Power interface reset
: RCC_APB1RSTR_DACRST   %1 29 lshift RCC_APB1RSTR bis! ;  \ RCC_APB1RSTR_DACRST    DAC interface reset
: RCC_APB1RSTR_CECRST   %1 30 lshift RCC_APB1RSTR bis! ;  \ RCC_APB1RSTR_CECRST    HDMI CEC reset

\ RCC_AHBENR (read-write)
: RCC_AHBENR_DMAEN   %1 0 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_DMAEN    DMA1 clock enable
: RCC_AHBENR_SRAMEN   %1 2 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_SRAMEN    SRAM interface clock  enable
: RCC_AHBENR_FLITFEN   %1 4 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_FLITFEN    FLITF clock enable
: RCC_AHBENR_CRCEN   %1 6 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_CRCEN    CRC clock enable
: RCC_AHBENR_IOPAEN   %1 17 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_IOPAEN    I/O port A clock enable
: RCC_AHBENR_IOPBEN   %1 18 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_IOPBEN    I/O port B clock enable
: RCC_AHBENR_IOPCEN   %1 19 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_IOPCEN    I/O port C clock enable
: RCC_AHBENR_IOPDEN   %1 20 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_IOPDEN    I/O port D clock enable
: RCC_AHBENR_IOPFEN   %1 22 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_IOPFEN    I/O port F clock enable
: RCC_AHBENR_TSCEN   %1 24 lshift RCC_AHBENR bis! ;  \ RCC_AHBENR_TSCEN    Touch sensing controller clock  enable

\ RCC_APB2ENR (read-write)
: RCC_APB2ENR_SYSCFGEN   %1 0 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_SYSCFGEN    SYSCFG clock enable
: RCC_APB2ENR_ADCEN   %1 9 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_ADCEN    ADC 1 interface clock  enable
: RCC_APB2ENR_TIM1EN   %1 11 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_TIM1EN    TIM1 Timer clock enable
: RCC_APB2ENR_SPI1EN   %1 12 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_SPI1EN    SPI 1 clock enable
: RCC_APB2ENR_USART1EN   %1 14 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_USART1EN    USART1 clock enable
: RCC_APB2ENR_TIM15EN   %1 16 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_TIM15EN    TIM15 timer clock enable
: RCC_APB2ENR_TIM16EN   %1 17 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_TIM16EN    TIM16 timer clock enable
: RCC_APB2ENR_TIM17EN   %1 18 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_TIM17EN    TIM17 timer clock enable
: RCC_APB2ENR_DBGMCUEN   %1 22 lshift RCC_APB2ENR bis! ;  \ RCC_APB2ENR_DBGMCUEN    MCU debug module clock  enable

\ RCC_APB1ENR (read-write)
: RCC_APB1ENR_TIM2EN   %1 0 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_TIM2EN    Timer 2 clock enable
: RCC_APB1ENR_TIM3EN   %1 1 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_TIM3EN    Timer 3 clock enable
: RCC_APB1ENR_TIM6EN   %1 4 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_TIM6EN    Timer 6 clock enable
: RCC_APB1ENR_TIM14EN   %1 8 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_TIM14EN    Timer 14 clock enable
: RCC_APB1ENR_WWDGEN   %1 11 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_WWDGEN    Window watchdog clock  enable
: RCC_APB1ENR_SPI2EN   %1 14 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_SPI2EN    SPI 2 clock enable
: RCC_APB1ENR_USART2EN   %1 17 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_USART2EN    USART 2 clock enable
: RCC_APB1ENR_I2C1EN   %1 21 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_I2C1EN    I2C 1 clock enable
: RCC_APB1ENR_I2C2EN   %1 22 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_I2C2EN    I2C 2 clock enable
: RCC_APB1ENR_PWREN   %1 28 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_PWREN    Power interface clock  enable
: RCC_APB1ENR_DACEN   %1 29 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_DACEN    DAC interface clock enable
: RCC_APB1ENR_CECEN   %1 30 lshift RCC_APB1ENR bis! ;  \ RCC_APB1ENR_CECEN    HDMI CEC interface clock  enable

\ RCC_BDCR ()
: RCC_BDCR_LSEON   %1 0 lshift RCC_BDCR bis! ;  \ RCC_BDCR_LSEON    External Low Speed oscillator  enable
: RCC_BDCR_LSERDY   %1 1 lshift RCC_BDCR bis! ;  \ RCC_BDCR_LSERDY    External Low Speed oscillator  ready
: RCC_BDCR_LSEBYP   %1 2 lshift RCC_BDCR bis! ;  \ RCC_BDCR_LSEBYP    External Low Speed oscillator  bypass
: RCC_BDCR_LSEDRV   ( %XX -- ) 3 lshift RCC_BDCR bis! ;  \ RCC_BDCR_LSEDRV    LSE oscillator drive  capability
: RCC_BDCR_RTCSEL   ( %XX -- ) 8 lshift RCC_BDCR bis! ;  \ RCC_BDCR_RTCSEL    RTC clock source selection
: RCC_BDCR_RTCEN   %1 15 lshift RCC_BDCR bis! ;  \ RCC_BDCR_RTCEN    RTC clock enable
: RCC_BDCR_BDRST   %1 16 lshift RCC_BDCR bis! ;  \ RCC_BDCR_BDRST    Backup domain software  reset

\ RCC_CSR ()
: RCC_CSR_LSION   %1 0 lshift RCC_CSR bis! ;  \ RCC_CSR_LSION    Internal low speed oscillator  enable
: RCC_CSR_LSIRDY   %1 1 lshift RCC_CSR bis! ;  \ RCC_CSR_LSIRDY    Internal low speed oscillator  ready
: RCC_CSR_RMVF   %1 24 lshift RCC_CSR bis! ;  \ RCC_CSR_RMVF    Remove reset flag
: RCC_CSR_OBLRSTF   %1 25 lshift RCC_CSR bis! ;  \ RCC_CSR_OBLRSTF    Option byte loader reset  flag
: RCC_CSR_PINRSTF   %1 26 lshift RCC_CSR bis! ;  \ RCC_CSR_PINRSTF    PIN reset flag
: RCC_CSR_PORRSTF   %1 27 lshift RCC_CSR bis! ;  \ RCC_CSR_PORRSTF    POR/PDR reset flag
: RCC_CSR_SFTRSTF   %1 28 lshift RCC_CSR bis! ;  \ RCC_CSR_SFTRSTF    Software reset flag
: RCC_CSR_IWDGRSTF   %1 29 lshift RCC_CSR bis! ;  \ RCC_CSR_IWDGRSTF    Independent watchdog reset  flag
: RCC_CSR_WWDGRSTF   %1 30 lshift RCC_CSR bis! ;  \ RCC_CSR_WWDGRSTF    Window watchdog reset flag
: RCC_CSR_LPWRRSTF   %1 31 lshift RCC_CSR bis! ;  \ RCC_CSR_LPWRRSTF    Low-power reset flag

\ RCC_AHBRSTR (read-write)
: RCC_AHBRSTR_IOPARST   %1 17 lshift RCC_AHBRSTR bis! ;  \ RCC_AHBRSTR_IOPARST    I/O port A reset
: RCC_AHBRSTR_IOPBRST   %1 18 lshift RCC_AHBRSTR bis! ;  \ RCC_AHBRSTR_IOPBRST    I/O port B reset
: RCC_AHBRSTR_IOPCRST   %1 19 lshift RCC_AHBRSTR bis! ;  \ RCC_AHBRSTR_IOPCRST    I/O port C reset
: RCC_AHBRSTR_IOPDRST   %1 20 lshift RCC_AHBRSTR bis! ;  \ RCC_AHBRSTR_IOPDRST    I/O port D reset
: RCC_AHBRSTR_IOPFRST   %1 22 lshift RCC_AHBRSTR bis! ;  \ RCC_AHBRSTR_IOPFRST    I/O port F reset
: RCC_AHBRSTR_TSCRST   %1 24 lshift RCC_AHBRSTR bis! ;  \ RCC_AHBRSTR_TSCRST    Touch sensing controller  reset

\ RCC_CFGR2 (read-write)
: RCC_CFGR2_PREDIV   ( %XXXX -- ) 0 lshift RCC_CFGR2 bis! ;  \ RCC_CFGR2_PREDIV    PREDIV division factor

\ RCC_CFGR3 (read-write)
: RCC_CFGR3_USART1SW   ( %XX -- ) 0 lshift RCC_CFGR3 bis! ;  \ RCC_CFGR3_USART1SW    USART1 clock source  selection
: RCC_CFGR3_I2C1SW   %1 4 lshift RCC_CFGR3 bis! ;  \ RCC_CFGR3_I2C1SW    I2C1 clock source  selection
: RCC_CFGR3_CECSW   %1 6 lshift RCC_CFGR3 bis! ;  \ RCC_CFGR3_CECSW    HDMI CEC clock source  selection
: RCC_CFGR3_ADCSW   %1 8 lshift RCC_CFGR3 bis! ;  \ RCC_CFGR3_ADCSW    ADC clock source selection

\ RCC_CR2 ()
: RCC_CR2_HSI14ON   %1 0 lshift RCC_CR2 bis! ;  \ RCC_CR2_HSI14ON    HSI14 clock enable
: RCC_CR2_HSI14RDY   %1 1 lshift RCC_CR2 bis! ;  \ RCC_CR2_HSI14RDY    HR14 clock ready flag
: RCC_CR2_HSI14DIS   %1 2 lshift RCC_CR2 bis! ;  \ RCC_CR2_HSI14DIS    HSI14 clock request from ADC  disable
bitfields.fs                                                        911,8          80%
-- INSERT --
```

feh [1 of 1] - f0-lmt01-temp-sensor.png
LMT01 Temperature Sensor Flowchart
<Terry Porter 2019, terry@tjporter.com.au>
A
START
word: lmt01-x.power.on
Power up LMT01-x
B1
Has lmt01 pulse been received ?
(via interrupt)
NO
YES
C
Start 105mS WINDOW timer
Tim6 via interrupt
D
COUNT LMT01 pulses
(via interrupt)
NO
E
Has 105mS WINDOW CLOSED ?
YES
word: lmt01-x.power.off
Power down LMT01-X
word: lmt01-fault?
Is COUNT > 0 ?
NO
YES
G
SENSOR FAULTY or
NOT PRESENT
word: degrees
Scale COUNT for Degrees
C & F
J
FAULT
I
END

Youtube Videos

https://www.youtube.com/watch?v=kDi-NIz3-QA

https://www.youtube.com/watch?v=7DfeEcU-tAs

YouTube Demo Code

```
\ Program Name: ut-f0-disco-demo.fs
\ Date: Thu 29 Nov 2018 16:57:34 AEDT
\ Copyright 2018  t.porter <terry@tjporter.com.au>, licensed under a BSD Licence.
\ For Mecrisp-Stellaris by Matthias Koch. /
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F051, Board: STM32F0 Discovery Board
\ Clock: 8 Mhz using the internal STM32F051 RC clock, unless otherwise stated
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\
\ This Program : A simple 'Blinky' demo for You-Tube
\
\  This System is designed for a modified
\  /mecrisp-stellaris/mecrisp-stellaris-patches-and-kernels/
\  discof0-rtscts-480600-nlcr-green-blue-pc7cerror.bin kernel
\  This kernel sets the two leds as outputs for 'blue-on', 'green-on'
\  and 'blue-off', 'green-off'
\
\  Multitasking is in use see /library/f0-multitask.fs which is loaded last
\  Add 'pause' in looping words for good multitasking response.
\
\  Design program then click on the Gvim 'make' button to upload this program
\  to the target chip and run it.
\ ---------------------------------------------------------------------------\

 compiletoram
 f0-disco-48mhz


 0 variable counter
 1 variable flag

 : delay ( -- )                                         \ Non Blocking Delay
    counter @ 100000 = if
                 0 counter !
                 1 flag @ xor flag !
                    else counter @ 1+ counter !
                 then ;

 : leds-on-or-off?
      flag @ 0 =  if green-on blue-on
                    else green-off blue-off
                 then ;

 : main
      pause                                             \ essential Word
      delay
      leds-on-or-off? ;


 : init-general                                         \ essential Multitasking and
Initialization Word
   f0-disco-48mhz ;
```

Want to know the meaning of the Words above ?

Historical Forth Development

Historically, Forth code was edited on a Host in a simple editor then uploaded to the Target Chip via a 9600 Baud serial terminal application such as Picocom or code was interactively entered directly into the serial terminal to interact in real time with the Target hardware. The basic design of Forth hasn’t changed much over the last 30 years and why would it?

Source-code uploads at 9600 Baud with only mk1-Eyeball code error checking was fine back in the day, but that day was 30 years ago.

One factor that still affects us today is that most Forths don’t have any communication handshaking between the terminal and the hardware, and where XON/XOFF software handshaking is available at one end, it must also be at the other. This situation is the same with hardware handshaking.

Note

Mecrisp-Stellaris does not come with communications handshaking either and the Picocom default serial terminal example syntax uses a end-of-line-delay to allow longer Words to be compiled without choking the on-chip compiler. This end-of-line-delay applies to all code including comments. I imagine this is a safe strategy so whilst comms may be slow, it doesn’t depend on a working handshaking configuration and the developer may later add handshaking if desired.

Modern Development Systems

Nowadays high level languages such as C running GGB via OpenOCD or SWD to the Target chip, are I believe, far more attractive to embedded developers than the historical Forth development, even if some of them such as the free Java based [atollic.studio] require a quad core PC with 32GB ram for a reasonable user experience.

C v/s Historical Forth

System	Language	Target Upload Speed	Debugging	Interactive	Complexity	Resources	Head Files
Historical Forth	Forth	9600 Baud	slow	yes	low	low	minir
Atollic Studio	C	2Mbits/s (swd)	fast	yes (GDB)	very high	quadcore PC	yes

Coming from a C background, when I started using Forth I found the historical methods far too slow and error-prone for my taste and over the last three years have slowly developed a more modern Forth IDE that I believe is as fast, or faster than the C alternatives.

C v/s Modern Forth

System	Language	Target Upload Speed	Debugging	Interactive	Complexity	Resources	Head Files
Modern Forth	Forth	460800 Baud	fast	yes	low	Low - RPI	yes (svd2f
Atollic Studio	C	2Mbits/s (swd)	fast	yes (GDB)	very high	quadcore pc	yes

Modern Forth IDE Elements

Description	Software	Reason for use
Serial terminal	Gnu Screen	Scriptable
Code editor	Vim-Gtk	Versatile and capable
Bourne Shell	SH	Common on all Unix systems

Description	**Software**	**Reason for use**
Coding/library support	sfv2forth	Automatically generate all Register and Bitfield memory mapped Words
Software Code Management	Fossil	Personal preference based on ease of use, capabilities and rock solid reliability.
Target Baud Rate increase to 460800	Mecrisp Kernel Mod 1	48x source-code upload speed increase
Hardware handshaking (RTS)	Mecrisp Kernel Mod 2	Prevent the on-chip Forth Compiler from choking up
Automatic source-code error detection	Mecrisp Kernel Mod 3	Stop the upload when a source-code error is encountered
Add CR to LF	Mecrisp Kernel Mod 4	Stop ref:*stair stepping<stair-stepping>* on Gnu Screen

Serial Terminal: GnuScreen

In recent times, Forth developers have built custom serial terminals which include methods to speed up the development process. Custom serial terminals sometimes have these drawbacks:

- are designed for a specific Forth and operating system
- dependencies that don't work properly on other operating systems
- closed source

GnuScreen

Warning

Without RTS handshaking, (even at only 115200 Baud) a STM32F0xx at 48Mhz will probably choke on some Forth source code, resulting in the terminal 'freezing' until the Target is rebooted. This unsatisfactory situation can be worked around (until you impliment RTS handshaking for your MCU) by using the GnuScreen 'slowpaste' option.

Without RTS Handshaking

Note

you will probably have to change the serial device name to something other than my FreeBSD “/dev/cuaU0”

```
xterm -fs 12 -title 'GnuScreen@ 115200 bps |  CTRL A :quit |  ' -geometry 82x72+0+0 -e screen
-t ProjectName -S ProjectName  /dev/cuaU0 115200 &
```

GnuScreen Slowpaste (msec)

GnuScreen has a Slow Paste Capabability. This adds a delay after each character in milliseconds.

- Either

Enter this at the GnuScreen Terminal before each file you want to upload to the target.

```
<ctrl A> slowpaste 50
```

- Or

Add this to your ~/.screenrc and make sure all instances of GnuScreen are killed before using it.

```
defslowpaste 50
```

With RTS Handshaking

Note

you will probably have to change the serial device name to something other than my FreeBSD “/dev/cuaU0”

```
xterm -fs 12 -title 'GnuScreen@ 115200 bps |  CTRL A :quit |  ' -geometry 82x72+0+0 -e screen
-t ProjectName -S ProjectName  /dev/cuaU0 115200,crtscts &
```

Speedup Methods Employed

Issue	Custom Serial Terminal Solution
end of line delay	Parse the Forth receive data and wait for “OK” before uploading the next line of code
error detection	Parse the Forth receive data and halt the source upload when one is encountered
comments	Strip comments from the source before uploading them to the Target.
source-code sequencing	The use of #include and #require commands inside Forth source-code files

Gnu Screen is unique in that it is easily scripted to provide serial communications. This means that it can be closely tied to a editor forming a kind of IDE freeing the programmer from entering/uploading/pasting source-code into a serial terminal and allowing the workflow to be editor based instead.

Issue	**Gnu-Screen Solution**
end of line delay	RTS Hardware handshaking
error detection	Hardware handshaking from the Target chip Forth compiler which generates the error messages in the first place.
comments	Strip comments from the source before uploading them to the Target.
source-code sequencing	Source-code file positioning in the Project script file

Automatic Forth Command and File Uploading/sequencing

```
load-memmap.sh ...............................

#!/bin/sh
screen -p tm4c1294 -X stuff 'compiletoflash\n'
screen -p tm4c1294 -X readreg e pll.fs
screen -p tm4c1294 -X paste  e
screen -p tm4c1294 -X stuff '120MHz\n'
screen -p tm4c1294 -X readreg f memmap.fs
screen -p tm4c1294 -X paste  f
screen -p tm4c1294 -X stuff 'compiletoram\n'

load-memmap.sh ...............................
```

When “load-memmap.sh: is executed, the command “compiletoflash” is sent to the Target (as if it were typed into the terminal by hand), instructing it to compile Forth source-code into the Flash memory.

The source-code file “pll.fs” is then loaded into ‘register a’ and ‘register a’ is then ‘pasted’ (uploaded) to the Target at 460800 Baud, which is too fast to read on the GnuScreen terminal. Note that pll.fs is loaded first as it’s a dependency for the next file to be loaded.

There are 26 ‘registers’ available (a - z) to store and paste files if required.

The Forth Word “120MHz” (provided by “pll.fs”) is then executed, and increases the Target clock from 16MHz to 120 MHz so everything now happens 7.5 times faster.

memmap.fs is then uploaded and the Forth on-chip compiler is then switched back to compiling source-code into RAM.

At this point I can manually enter a command in the Gnu-Screen terminal such as “rcc.” and read the real time values in register ‘rcc’ on the Target chip itself.

```
rcc. RCC_CR () $00005383
            P             H H
            L P         C S S H                         H
            L L         S E E S                         S H
            R L         S B R E                         I S
            D O         O Y D O|    HSICAL     |         |  R I
            Y N         N P Y N|7:6:5:4:3:2:1:0| HSITRIM |  D O
           |2|2|       |1|1|1|1|1|1|1|1|1|1| | |4:3:2:1:0|  Y N
~|~|~|~|~|~|5|4|~|~|~|~|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|~|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 1 1
```

Note

This scripting ability allows me to sequence the project build operations into a Makefile, so that when I click on the “Make” icon in the VIM-GTK toolbar, the upload and error checking are automatic just like in a modern C IDE tied to the Target via OpenOCD, Jtag, GDB or whatever.

Note

The creation and sequencing of project files is not done by hand, it’s done automatically by a Project Builder (written in SH of course) which takes the name of a new project and then builds everything needed into a new directory of the same name.

Comment and Space Removal (RCAS)

Whilst extensive Forth source-code comments are vital for design and maintenance and blank lines aid readability, they mean nothing to the on-chip compiler, so before uploading the source-code file, we can strip all comments and empty lines. The source-code file is unaltered and upload time is decreased.

RCAS script code

Usage: rcas.sh <file>

```
# rcas.sh ................cut here....................

sed 's/\\.*$//g' < $1 |  sed '/^\s*$/d' > $1.rcas.fs

# rcas.sh ................cut here....................
```

Rcas example with Gnu Screen

```
FILE_NAME=forthcode.fs

screen -p rcas -X readreg x `./rcas.sh $FILE_NAME` $FILE_NAME.rcas.fs
screen -p rcas -X paste x
rm $FILE_NAME.rcas.fs
```

Code editor

Vim-GTK or ‘gvim’ leverages the VIM editor and the GTK widget set to create a useful GUI editor. Of course any other editor with similar flexibility can be used.

Bourne Shell

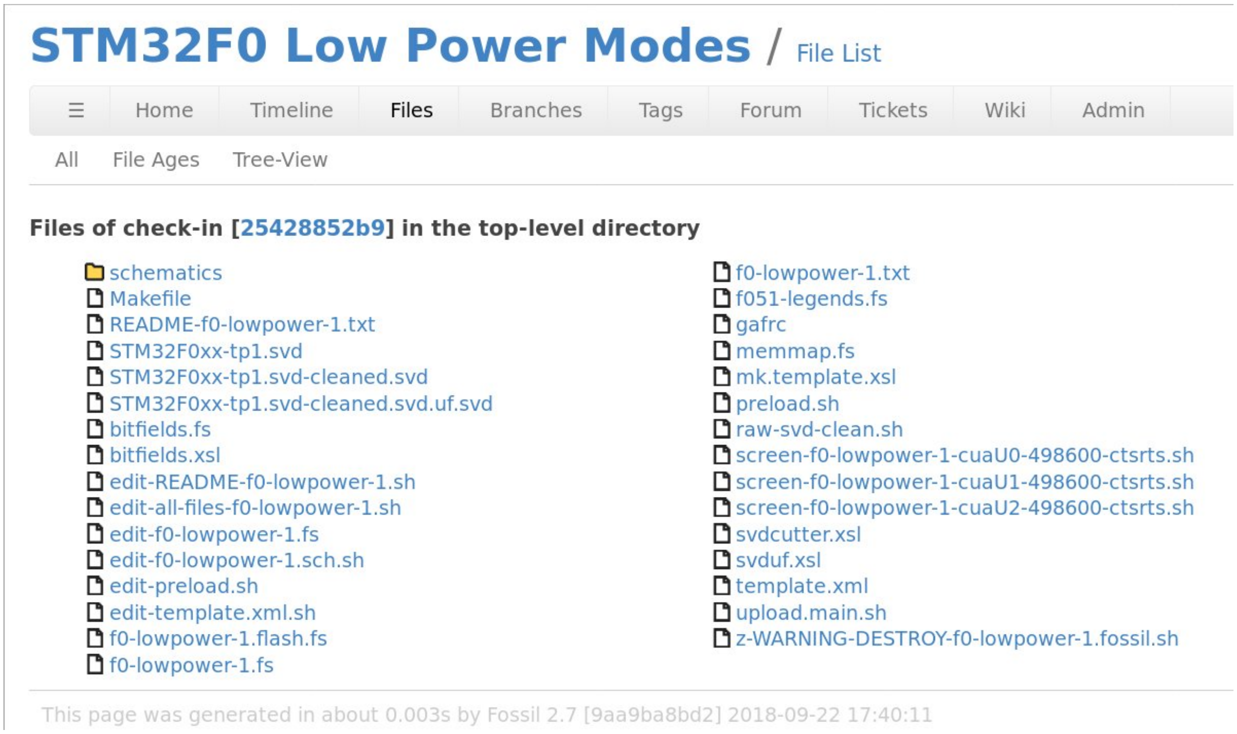
Unlike Bash which is so common on Linux systems, the Bourne Shell (sh) is present of every Unix I have seen, so it’s a great compitability choice to provide the glue to tie everything together without needing to install yet another application.

Coding/library support

Modern Microprocessors such as ARM variants have a lot of peripherals, registers and register bit-fields, far too many for any coder to ever create memory mapped Words by hand. I wrote svd2forth to automatically derive Forth Words for Mecrisp-Stellaris from CMSIS-SVD system used by ARM, and this also forms part of the IDE.

Fossil SCM

Fossil is also integrated into my Project Builder so the new project automatically has its own SCM.



STM32F0 Low Power Modes / File List

Files of check-in [25428852b9] in the top-level directory

- schematics
- Makefile
- README-f0-lowpower-1.txt
- STM32F0xx-tp1.svd
- STM32F0xx-tp1.svd-cleaned.svd
- STM32F0xx-tp1.svd-cleaned.svd.uf.svd
- bitfields.fs
- bitfields.xsl
- edit-README-f0-lowpower-1.sh
- edit-all-files-f0-lowpower-1.sh
- edit-f0-lowpower-1.fs
- edit-f0-lowpower-1.sch.sh
- edit-preload.sh
- edit-template.xml.sh
- f0-lowpower-1.flash.fs
- f0-lowpower-1.fs
- f0-lowpower-1.txt
- f051-legends.fs
- gafrc
- memmap.fs
- mk.template.xsl
- preload.sh
- raw-svd-clean.sh
- screen-f0-lowpower-1-cuaU0-498600-ctsrts.sh
- screen-f0-lowpower-1-cuaU1-498600-ctsrts.sh
- screen-f0-lowpower-1-cuaU2-498600-ctsrts.sh
- svdcutter.xsl
- svduf.xsl
- template.xml
- upload.main.sh
- z-WARNING-DESTROY-f0-lowpower-1.fossil.sh

This page was generated in about 0.003s by Fossil 2.7 [9aa9ba8bd2] 2018-09-22 17:40:11

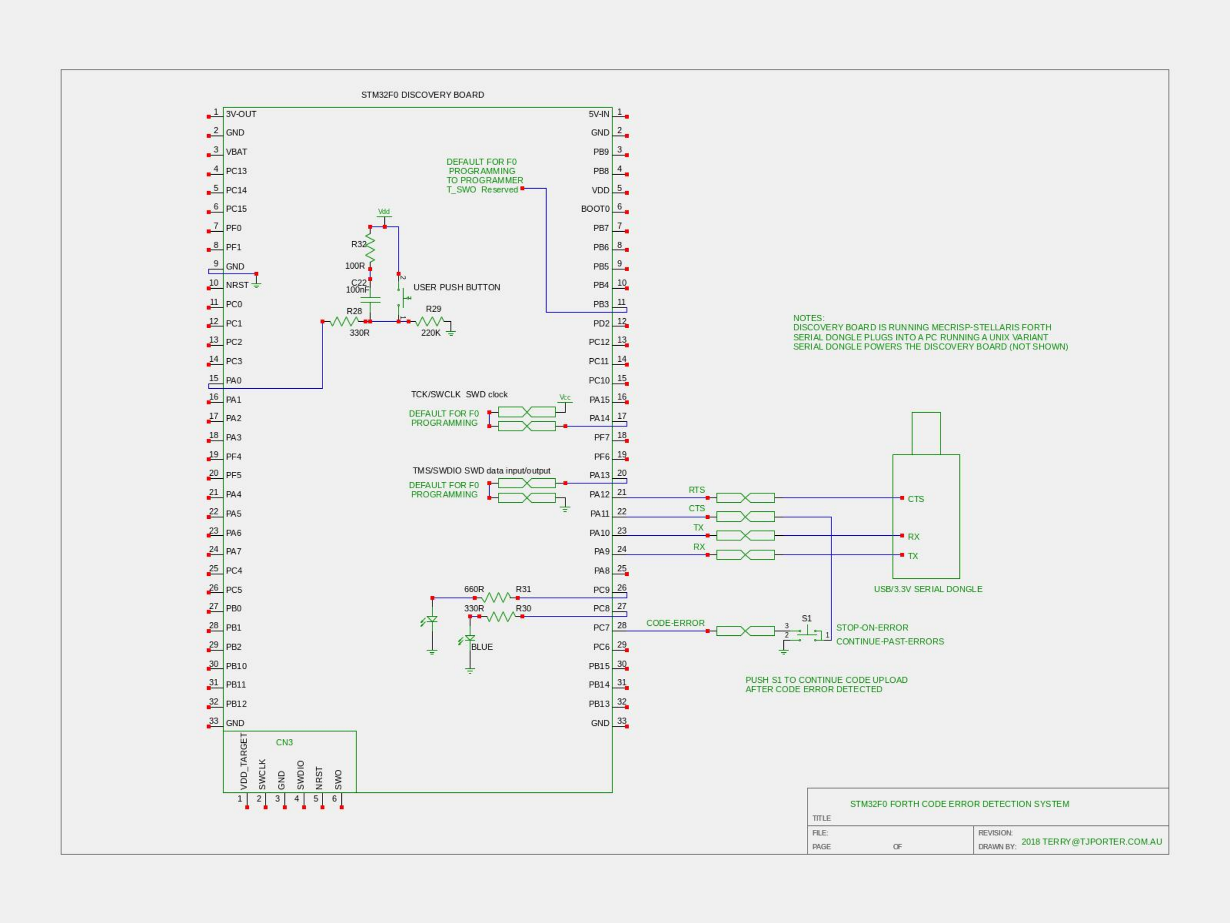
Target Baud Rate increase to 480600

A simple Assembly Language Kernel hack to terminal.s in the Target chip directory of Mecrisp-Stellaris. This is nicely commented by Matthias, so just adjust the default value which provides 115200 Baud.

Hardware handshaking (RTS)

A simple Assembly Language Kernel hack to terminal.s in the Target chip directory of Mecrisp-Stellaris. This enables RTS hardware handshaking on USART1

Forth on-chip Compiler Hardware Handshaking

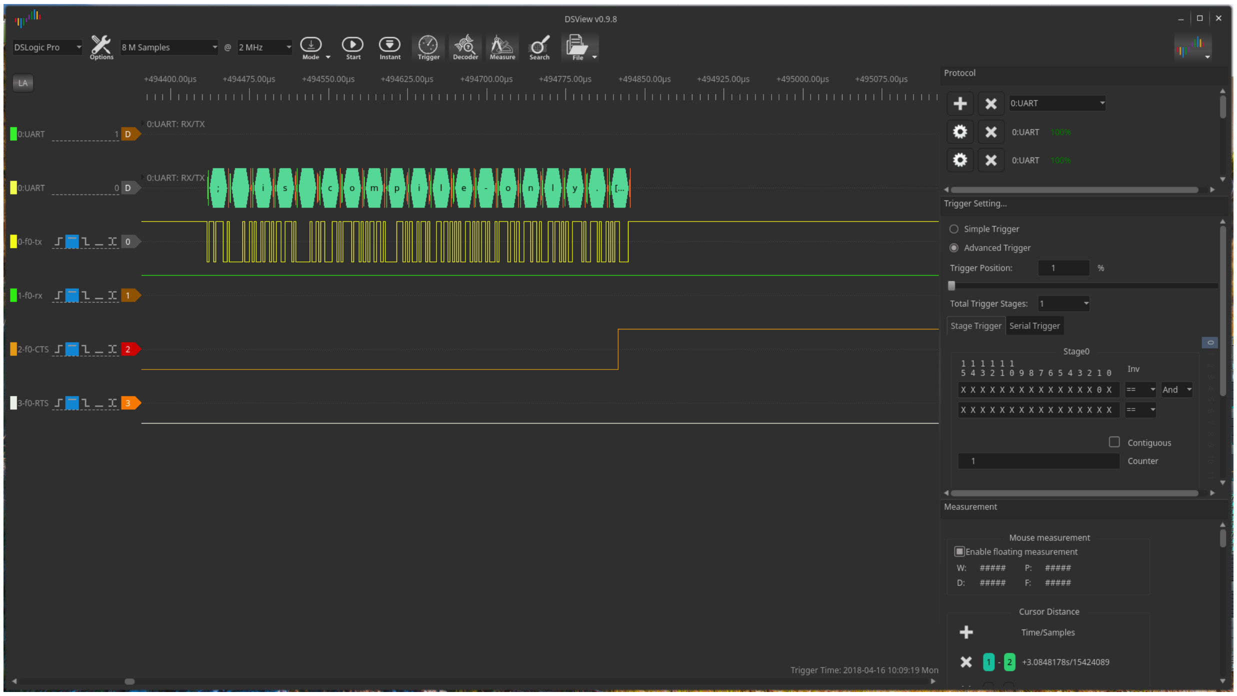

STM32F0 DISCOVERY BOARD
3V-OUT
GND
VBAT
PC13
PC14
PC15
PF0
PF1
GND
NRST
PC0
PC1
PC2
PC3
PA0
PA1
PA2
PA3
PF4
PF5
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PB10
PB11
PB12
GND
5V-IN
GND
PB9
PB8
VDD
BOOT0
PB7
PB6
PB5
PB4
PB3
PD2
PC12
PC11
PC10
PA15
PA14
PF7
PF6
PA13
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
GND
DEFAULT FOR F0 PROGRAMMING TO PROGRAMMER T_SWO Reserved
Vdd
R32
100R
C22
100nF
USER PUSH BUTTON
R28
330R
R29
220K
TCK/SWCLK SWD clock
DEFAULT FOR F0 PROGRAMMING
Vcc
TMS/SWDIO SWD data input/output
DEFAULT FOR F0 PROGRAMMING
RTS
CTS
TX
RX
CTS
RX
TX
USB/3.3V SERIAL DONGLE
NOTES:
DISCOVERY BOARD IS RUNNING MECRISP-STELLARIS FORTH
SERIAL DONGLE PLUGS INTO A PC RUNNING A UNIX VARIANT
SERIAL DONGLE POWERS THE DISCOVERY BOARD (NOT SHOWN)
660R
R31
330R
R30
BLUE
CODE-ERROR
S1
STOP-ON-ERROR
CONTINUE-PAST-ERRORS
PUSH S1 TO CONTINUE CODE UPLOAD AFTER CODE ERROR DETECTED
CN3
VDD_TARGET
SWCLK
GND
SWDIO
NRST
SWO
STM32F0 FORTH CODE ERROR DETECTION SYSTEM
TITLE
FILE:
PAGE
OF
REVISION:
DRAWN BY:
2018 TERRY@TJPORTER.COM.AU

Automatic source-code error detection

Assembly Language addition to the Mecrisp-Stellaris source.

Logic analyser showing the Forth on-chip compiler CTS signaling on error

This shows the Forth on-chip compiler announcing a source-code error “is compile only” has been detected, then raising PC-7 which is connected to CTS. This halts serial communications with the PC at the actual source-code error.


DSView v0.9.8
DSLogic Pro
8 M Samples
2 MHz
Options
Mode
Start
Instant
Trigger
Decoder
Measure
Search
File
0:UART: RX/TX
0:UART
0-f0-tx
1-f0-rx
2-f0-CTS
3-f0-RTS
; i s c o m p i l e - o n l y . [...
Protocol
Trigger Setting...
Simple Trigger
Advanced Trigger
Trigger Position:
Total Trigger Stages:
Stage Trigger
Serial Trigger
Stage0
Contiguous
Counter
Measurement
Mouse measurement
Enable floating measurement
Cursor Distance
Time/Samples
+3.0848178s/15424089
Trigger Time: 2018-04-16 10:09:19 Mon

Add CR to LF

Assembly Language addition to the Mecrisp-Stellaris source.

For some reason Gnu-Screen (at least on my FreeBSD system) seems to ignore any configuration to stop ‘stair stepping’. The received data from the Forth system only contains a (Unix) line feed (LF) at the end of each line, but Gnu-Screen wants a DOS (LF/CR) pair. Rather than try and hack the Gnu-Screen source, I just added a carriage return (CR) to the existing (LF) in the Mecrisp-Stellaris Kernel.

To prevent ‘stair stepping’ when using GnuScreen on your version of Mecrisp-Stellaris Forth, grep the Mecrisp-Stellaris datastackandmacros.s file (it’s in mecrisp-source/common/) for occurrences of

```
\Meldung\n
```

and replace them with

```
\Meldung\r\n
```

then rebuild the Mecrisp-Stellaris kernel and the GNUScreen stair stepping will be gone.

See also

Stair Stepping Page

MSP340

Mecrisp is for MSP430xx chips and not for Cortex-M like Mecrisp-Stellaris, nevertheless they are both written by Matthias Koch and have many similarities as far as Forth is concerned, but the hardware is totally different. Both Mecrisp and Mecrisp Stellaris are hosted on the same site.

See: https://sourceforge.net/projects/mecrisp/files/ for downloads.

Svd2Mecrisp

Svd2mecrisp was created to take advantage of SVD files produced by Vadzim Dambrouski's https://github.com/pftbest/msp430_svd which uses DSLITE files released by Ti and processes them with the Rust programming language.

Some features:- * Produces a Memory Map file you can upload via the Mecrisp serial terminal * Creates a Register-Reference file where you can copy and paste Bitfields you need into your program. Search this file using names from the Ti PDF's for rapid location and use. * Includes helpful longer descriptions for Registers and Bitfields for less PDF reliance.

Download Svd2mecrisp HERE. Instructions are included along with a sample msp430g2553.svd.

Sample Svd2Mecrisp output

Memory Map File

Partial, example only

```
...
\ ***** ADC10 ***** \
$48  constant  ADC10DTC0
$49  constant  ADC10DTC1
$4A  constant  ADC10AE0
$1B0  constant  ADC10CTL0
$1B2  constant  ADC10CTL1
$1B4  constant  ADC10MEM
$1BC  constant  ADC10SA
...
```

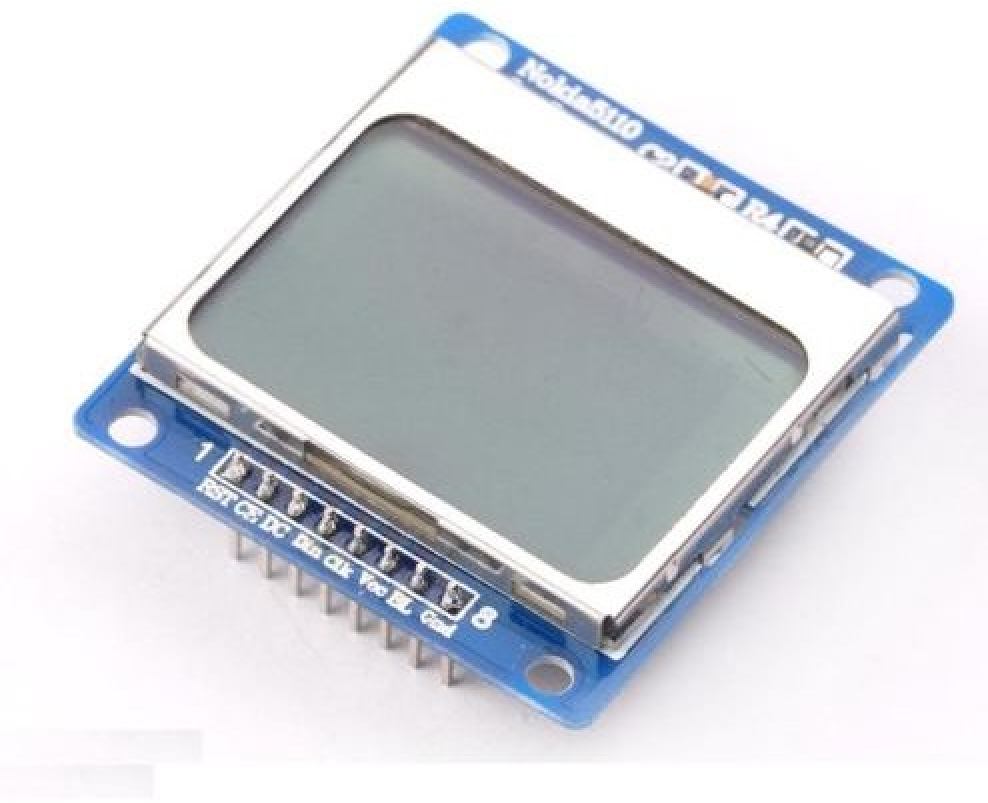
Register Reference File

Partial, example only

```
...
\ ADC10CTL0 ( ADC10 Control 0 )

\ %1  0 lshift constant ADC10SC     ( ADC10 Start Conversion
Bit 0 Width 1 )

\ %1  1 lshift constant ENC     ( ADC10 Enable Conversion
Bit 1 Width 1 )

\ %1  2 lshift constant ADC10IFG     ( ADC10 Interrupt Flag
Bit 2 Width 1 )

\ %1  3 lshift constant ADC10IE     ( ADC10 Interrupt Enalbe
Bit 3 Width 1 )

\ %1  4 lshift constant ADC10ON     ( ADC10 On/Enable   Bit 4
Width 1 )

\ %1  5 lshift constant REFON     ( ADC10 Reference on   Bit
5 Width 1 )

\ %1  6 lshift constant REF2_5V     ( ADC10 Ref 0:1.5V /
1:2.5V   Bit 6 Width 1 )

\ %1  7 lshift constant MSC     ( ADC10 Multiple
SampleConversion   Bit 7 Width 1 )
```

```
\ %1  8 lshift constant REFBURST     ( ADC10 Reference Burst
Mode   Bit 8 Width 1 )

\ %1  9 lshift constant REFOUT     ( ADC10 Enalbe output of
Ref.   Bit 9 Width 1 )

\ %1  10 lshift constant ADC10SR     ( ADC10 Sampling Rate
0:200ksps / 1:50ksps   Bit 10 Width 1 )

\ %xx  11 lshift constant ADC10SHT     ( ADC10 Sample Hold
Select Bit: 0   Bit 11 Width 2 )
    \ %0: ADC10SHT_0      ( 4 x ADC10CLKs )
    \ %1: ADC10SHT_1      ( 8 x ADC10CLKs )
    \ %11: ADC10SHT_2      ( 16 x ADC10CLKs )
    \ %111: ADC10SHT_3      ( 64 x ADC10CLKs )

\ %xxx  13 lshift constant SREF     ( ADC10 Reference Select
Bit: 0   Bit 13 Width 3 )
    \ %0: SREF_0      ( VR+ = AVCC and VR- = AVSS )
    \ %1: SREF_1      ( VR+ = VREF+ and VR- = AVSS )
    \ %11: SREF_2      ( VR+ = VEREF+ and VR- = AVSS )
    \ %111: SREF_3      ( VR+ = VEREF+ and VR- = AVSS )
    \ %1111: SREF_4      ( VR+ = AVCC and VR- = VREF-/VEREF-
)
    \ %11111: SREF_5      ( VR+ = VREF+ and VR- =
VREF-/VEREF- )
    \ %111111: SREF_6      ( VR+ = VEREF+ and VR- =
VREF-/VEREF- )
    \ %1111111: SREF_7      ( VR+ = VEREF+ and VR- =
VREF-/VEREF- )
...
```

Nokia 5110 LCD Display



Description

This is a very cheap and easy to use outdoor visible LCD display with led backlight, sourced mainly from recycled Nokia 5110 dumb phones. Many units such as the one pictured above (Banggood) have some small scratches from their previous lives, but the under $2 price makes them very popular for hobbyists. Also available from https://www.sparkfun.com

- Price Under $2 each (Banggood, Ebay). More from Sparkfun, but perhaps their displays are not scratched ?
- Operating voltage: 3 - 5v
- Resolution: 48 rows and 84 columns
- Operating Current without backlight: ≤ 200 µA
- Blue backlight
- Uses the Philips PCD8544 graphics controller chip.
- 3v logic, Google if you want to use 5v logic from ancient Arduinos

Documentation

- Nokia 5110 display: https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf
- Philips PCD8544 graphics controller chip: https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf

Synopsis

In this page I describe how to get a Nokia 5110 lcd display up and running, at least to test stage using Mecrisp-Stellaris, with code and Logic Analyser pictures of the data and control signals. This is for completeness as it’s actually very easy to get it working once you know what needs to be done. The trick is acquiring accurate knowledge about this display.

Even if you don’t use Forth, this may assist you the first time you run one of these dispays up.

Research

Whilst there are a great many YouTube articles regarding the 5110 lcd, the bulk are for Arduino in C, and short of many critical details, given the fondness of Arduino users for recycling libraries made by others. A special thank you to Julian Ilet for his excellent 5110 youtube videos: https://www.youtube.com/watch?v=RAlZ1DHw03g, without which I’d have taken considerably more time to get to this point.

Software Design

- Create and send the correct 5110 initialization signals
- Bit Bang serial commands to the display using a software SPI

Note

The 5110 requires the ‘SPI’ protocol of DATA and CLOCK. DATA is clocked in on the RISING EDGE of the CLOCK signal, so if the DATA line is LOW on the RISING CLOCK edge, then a DATA LOW is clocked in. The Bit Banging is done in code, and is quote simple.

Hardware and Software Used

- FreeBSD 11.1 OS, Linux will work fine also
- STM32F051, 32 bit CortexM based as standard in a STM32F0 Discovery Board
- Banggood model SKU078678 (2014) using a second hand Nokia 5110 LCD display.
- Open Bench Logic Sniffer http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer. A $5 Chinese Saleae logic 8 channel clone would be fine as well.

- Pulseview and Sigrok FLOSS logic analyser software https://sigrok.org/wiki/Main_Page

Pinouts and Signals

5110 Display pinouts and STM32F051 GPIO’s used

PIN	FUNCTION	GPIO	WIRE COLOR	NOTES
1	RST	PB2	ORANGE	Reset, active LOW. Internally this is the junction of a 10k pull up and a 104 cap to 0v.
2	CE	PB3	YELLOW	Chip Enable, active LOW
3	DC	PB4	GREEN	D=DATA, C=COMMAND Mode. Whe HIGH, you’re in DATA mode, LOW and you’re in COMMAND mode.
4	DATA	PB5	BLUE	
5	CLK	PB6	PURPLE	Active rising edge

PIN	FUNCTION	GPIO	WIRE COLOR	NOTES
6	VCC	3v	GREY	From the ‘3v’ pin on the STM32F0 Discovery Board
7	LIGHT	PB7	WHITE	Back Light, Active *HIGH*.
8	GND	0v	BLACK	

The very *First* Test

Make sure the LED Backlights work. If you have a different type of 5110 lcd display, your BL (backlight) pin may need to be LOW to work, unlike mine which are HIGH.

5110 *Essential* Initialization

Warning

Without the following sequence, the 5110 display will never display anything.

LCD Init commands

Warning

Sequence order must be followed top to bottom.

Hex	Name	Description
n/a	Rst	Low going pulse of about 1.7 milli seconds
$21	Extended	*serial command*: Lcd extended command mode
$B6	Contrast	*serial command*: Lcd set contrast command
4	Coeficient	*serial command*: Lcd set temperature coeficient command
$14	Bias	*serial command*: Lcd set bias command
$20	Basic	*serial command*: Lcd basic command mode

Logic Timing Diagram

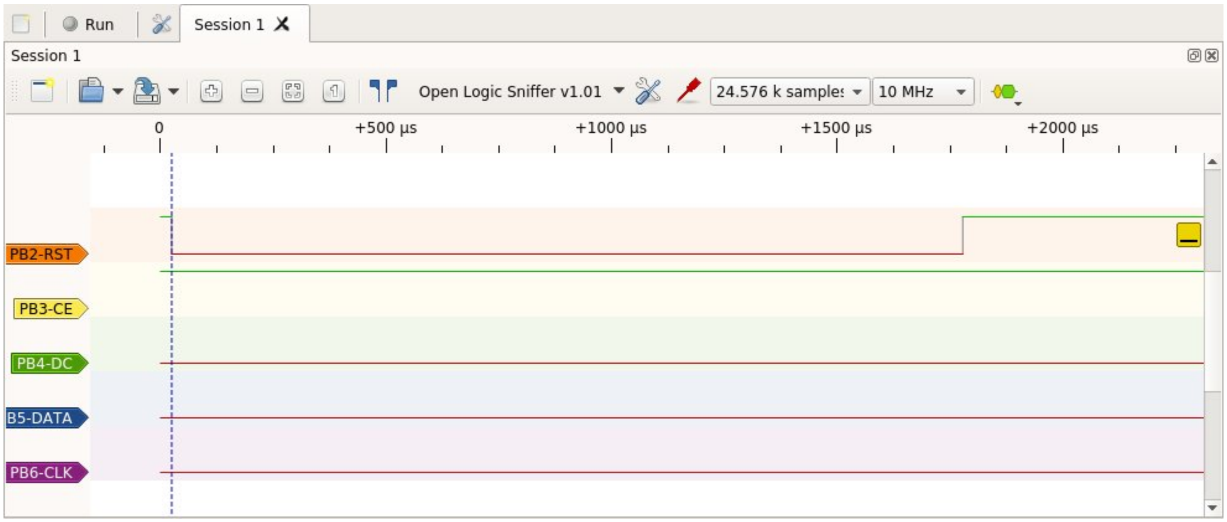
These are what I really wanted when I started the project because a picture is worth a thousand words, expecially when signal timing is unclear.

- Vcc is 3v and has been on a long time before any of the signals below are sent
- All serial commands consist of data and a clock, and are sent Most Significant Bit (MSB) *first*

- The Logic Analyser is triggered when CE (Chip Enable, active LOW) goes LOW and is signified by the dashed blue vertical line on each picture.

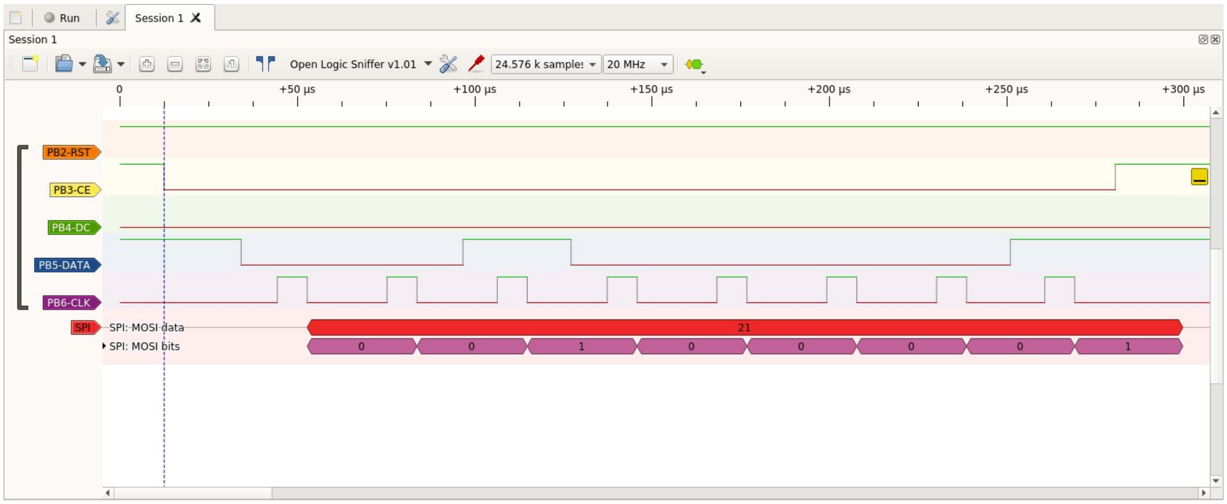
Lcd Reset

Could be shorter than 1.7 milli seconds I imagine, but realy short, i.e. with zero delay will not start the display.



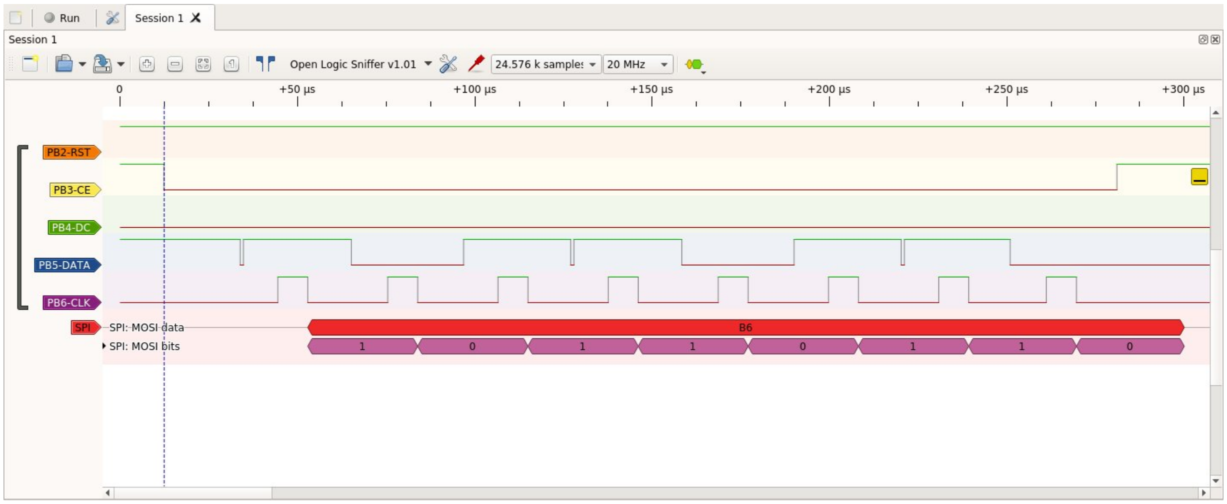
Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
10 MHz
0
+500 µs
+1000 µs
+1500 µs
+2000 µs
PB2-RST
PB3-CE
PB4-DC
B5-DATA
PB6-CLK

Extended Command ($21)

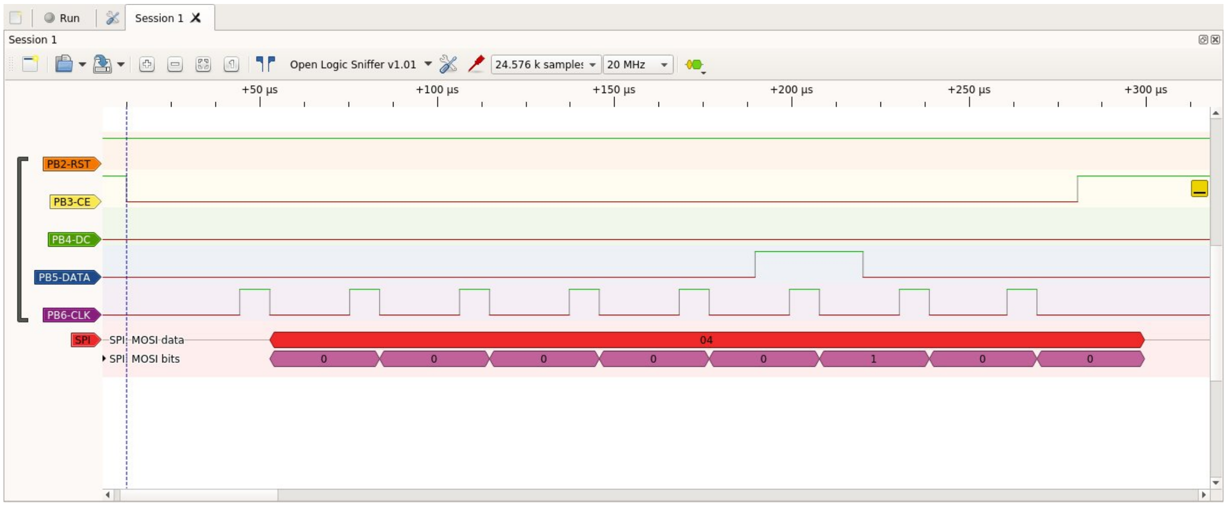


Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
20 MHz
0
+50 µs
+100 µs
+150 µs
+200 µs
+250 µs
+300 µs
PB2-RST
PB3-CE
PB4-DC
PB5-DATA
PB6-CLK
SPI
SPI: MOSI data
SPI: MOSI bits
21
0
0
1
0
0
0
0
1

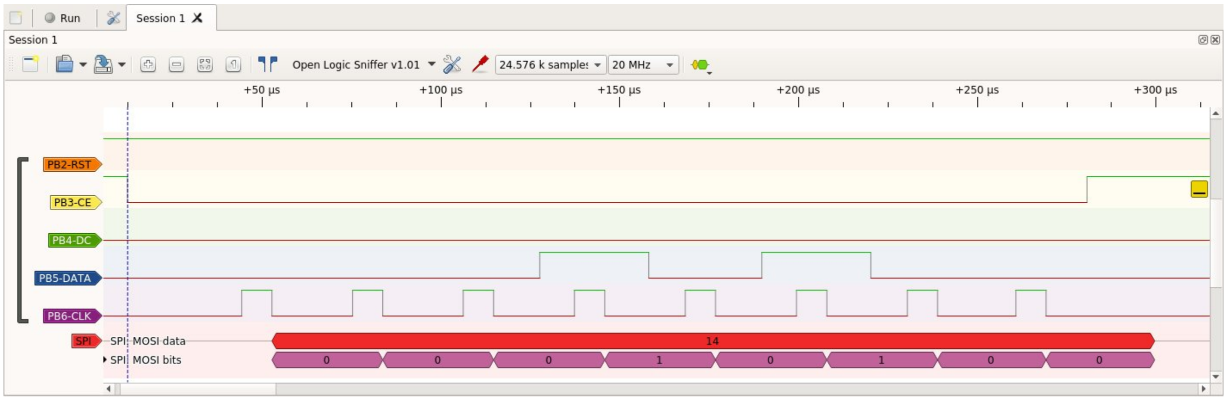
Contrast Command ($B6)


Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
20 MHz
0
+50 µs
+100 µs
+150 µs
+200 µs
+250 µs
+300 µs
PB2-RST
PB3-CE
PB4-DC
PB5-DATA
PB6-CLK
SPI
SPI: MOSI data
SPI: MOSI bits
B6
1
0
1
1
0
1
1
0

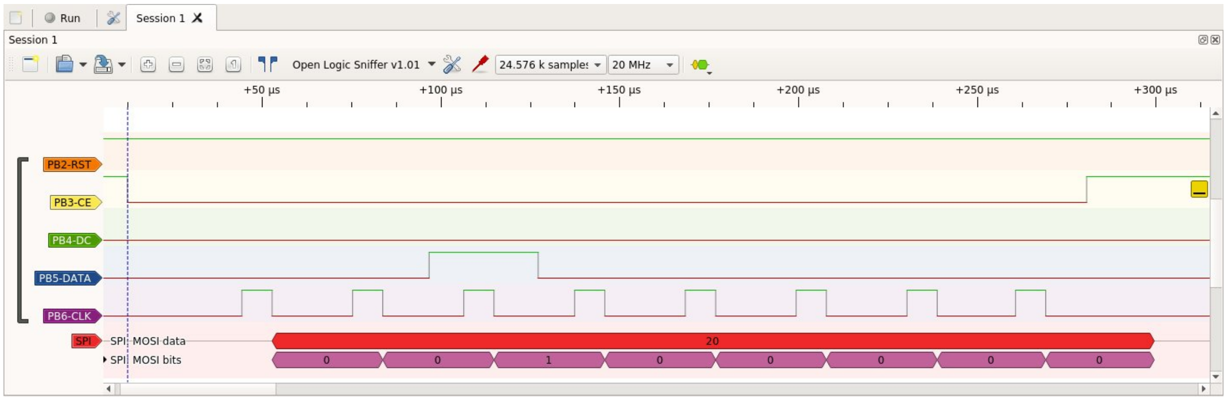
Temperature Coefficient Command(4)


Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
20 MHz
+50 µs
+100 µs
+150 µs
+200 µs
+250 µs
+300 µs
PB2-RST
PB3-CE
PB4-DC
PB5-DATA
PB6-CLK
SPI
SPI: MOSI data
SPI: MOSI bits
04
0
0
0
0
0
1
0
0

Bias Command ($14)


Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
20 MHz
+50 µs
+100 µs
+150 µs
+200 µs
+250 µs
+300 µs
PB2-RST
PB3-CE
PB4-DC
PB5-DATA
PB6-CLK
SPI
SPI: MOSI data
SPI: MOSI bits
14
0
0
0
1
0
1
0
0

Basic Command ($20)


Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
20 MHz
+50 µs
+100 µs
+150 µs
+200 µs
+250 µs
+300 µs
PB2-RST
PB3-CE
PB4-DC
PB5-DATA
PB6-CLK
SPI
SPI: MOSI data
SPI: MOSI bits
20
0
0
1
0
0
0
0
0

Note

This concludes the basic Init process. After the next command is run, all of the Pixels should be on.

All Pixels On Command (9)

Note

There are many kinds of Nokia 5110 Lcd displays on eBay, and some require different BIAS and COEFFICIENT values. If your display is still blank after running the commands above, I recommend you search online for settings known to work with your display.

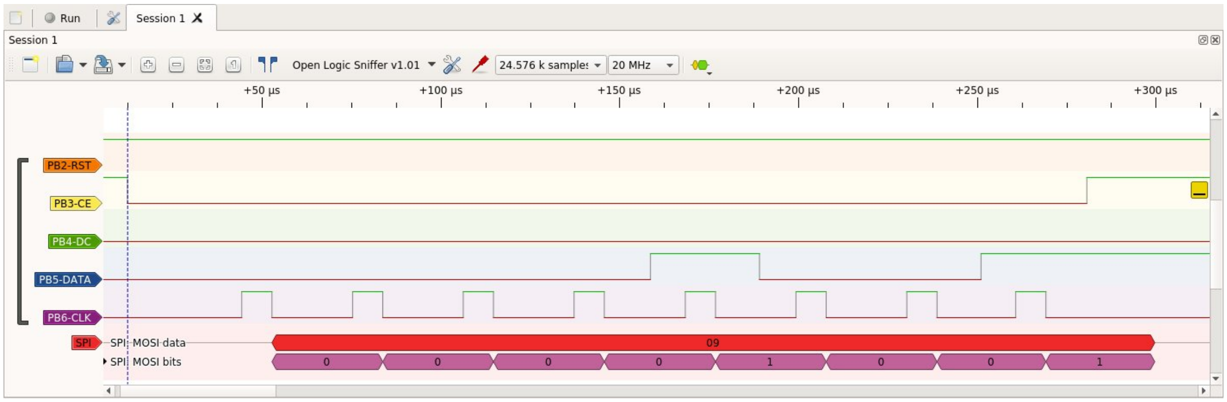
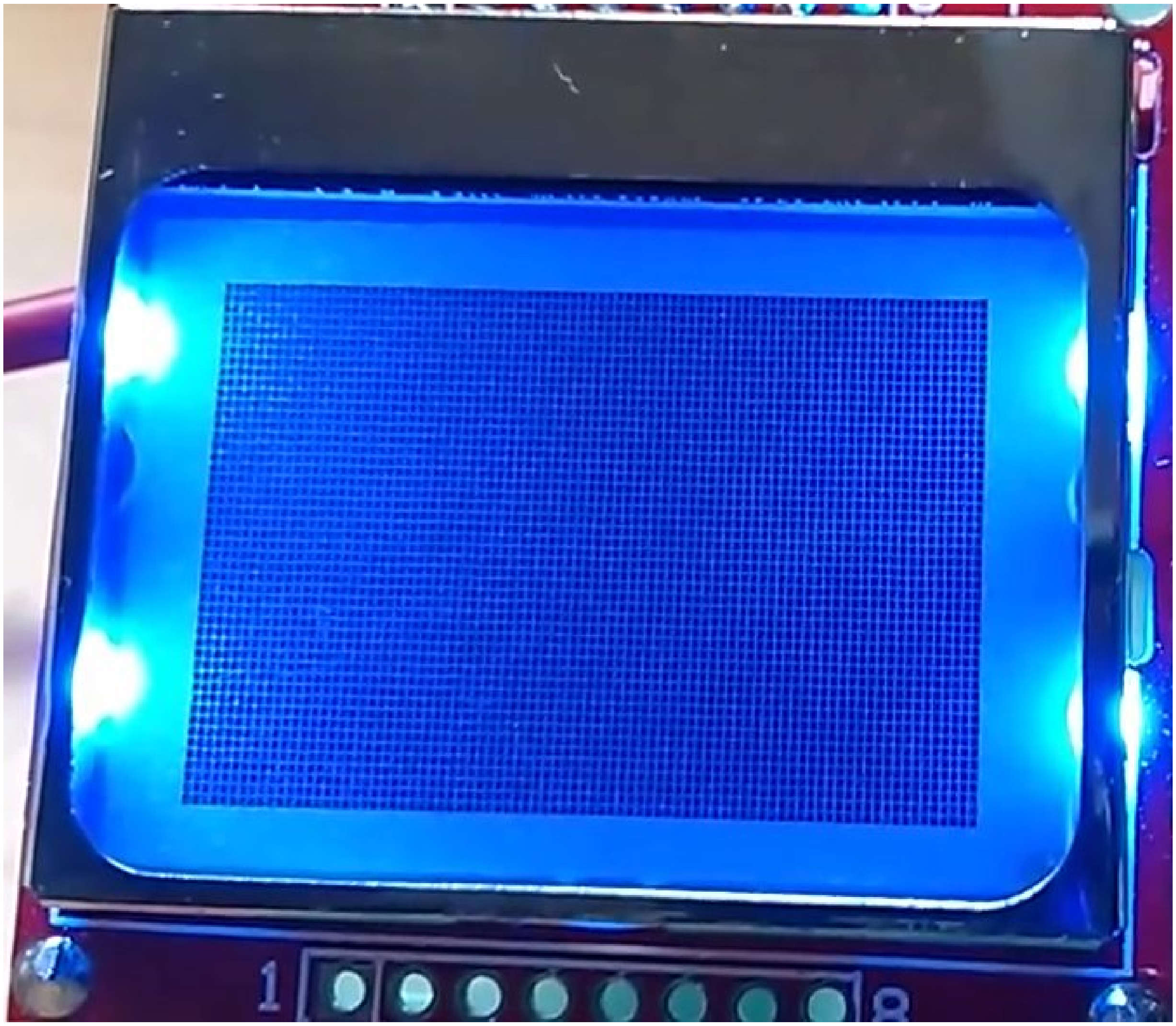

Run
Session 1
Session 1
Open Logic Sniffer v1.01
24.576 k samples
20 MHz
+50 µs
+100 µs
+150 µs
+200 µs
+250 µs
+300 µs
PB2-RST
PB3-CE
PB4-DC
PB5-DATA
PB6-CLK
SPI
SPI: MOSI data
SPI: MOSI bits
09
0
0
0
0
1
0
0
1

Photo of all pixels on

From the Julian Ilet video linked above.

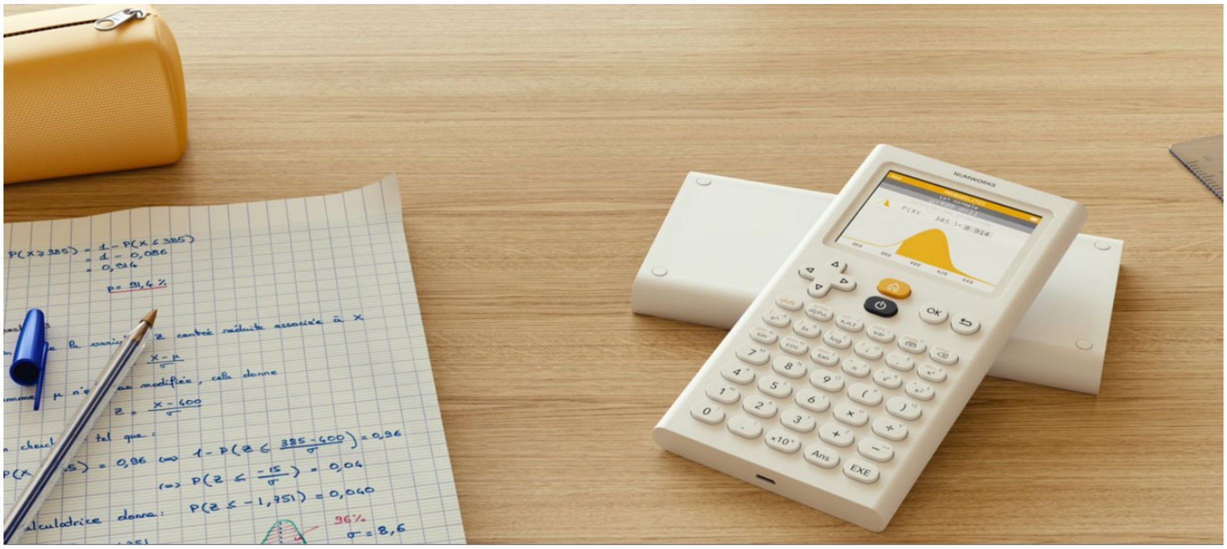


Forth Code

- Standalone, no other files needed. Just upload to a STM32F0 Discovery Board connected to a 5110 LCD.
- Will turn on All Pixels

`DOWNLOAD f052-nokia-5110-lcd.fs`

Numworks Calculator



https://www.numworks.com/simulator/

I haven’t been this excited about a calculator since I first drooled over pictures of the new HP35 (that I couldn’t afford) 35 years ago!

Appearing in release 2.4.7; Forth drivers by Wolfgang Strauss for the brilliant Open Source Numworks Calculator.

Is this the words first *real RPN calculator* ? ;-)

```
mecrisp-stellaris-2.4.7/numworks-ra
.
├── README
├── completesources.txt
├── cornerstone.txt
├── delay.txt
├── display.txt
├── editor.txt
├── epsilon.bin
├── graphics-unicode-8x16.txt
├── io.txt
├── keyboard.txt
├── mecrisp-stellaris-numworks-with-sources.bin
├── mecrisp-stellaris-numworks-with-sources.hex
├── mecrisp-stellaris-numworks.bin
├── multitask-debug.txt
├── release
└── terminal
```

Readme

```
-----------------------------------------------------------------------------------

Flashing of Mecrisp-Stellaris to the numworks calculator:

dfu-util -i 0 -a 0 -s 0x08000000 -D mecrisp-stellaris-numworks-with-sources.bin

No need to disassemble your calculator, terminal is on USART6: PA11 = D- = TX. PA12 = D+ = RX.
Just cut an old USB cable and connect it to a USB-serial-brige with 115200 Baud 8N1.

You need to type "eraseflash" once to clear the free dictionary space after you flash the Forth binary,
as dfu-util leaves old contents in flash blocks which are not used.

-----------------------------------------------------------------------------------

Do not forget to flash back to Epsilon at the end of the day or connect to a power supply,
as the Forth binary will drain your battery as long as sleep modes are not implemented:

dfu-util -i 0 -a 0 -s 0x08000000 -D epsilon.bin

Epsilon 1.6.0 binary included for convenience purposes only.
Compiled from sources found here: https://github.com/numworks/epsilon/
Unlike Mecrisp-Stellaris, Epsilon is NOT licensed under GPL3 !
```

```
------------------------------------------------------------
------------------
```

Patches: Kernel

1. Apply the patch Apply the patch
2. Compile Mecrisp-Stellaris

Testing A Patch Before Applying

```
patch --dry-run  < myfile.patch

mecrisp-stellaris-2.3.7/mecrisp-stellaris-source/common%
patch --dry-run < simpler-stackprint-deepinsight.s.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|--- deepinsight.s      2017-05-10 13:00:55.263208835 +1000
|+++ deepinsight-simplestackprint.s     2017-04-28 09:30:54.000000000 +1000
--------------------------
Patching file deepinsight.s using Plan A...
Hunk #1 succeeded at 87.
Hunk #2 succeeded at 116.
done
```

Applying patches

```
patch < myfile.patch
```

Reversing Patches

```
patch -R < myfile.patch
```

Create a Patch

```
diff -u oldfile newfile > myfile.patch
```

Patch: Simpler Stack Printer

Reason: While some write new Words to modify the default Stack printer “.s”, I prefer to patch the assembly source code.

```
From :          Stack: [7 ] 42 1 2 3 4 5 6  TOS: 7  \*>
To:             .s [7 ] 42 1 2 3 4 5 6 7  ok.

* Applies to: mecrisp-stellaris-2.3.7/mecrisp-stellaris-
source/common/deepinsight.s
* :download:`Simpler Stack Printer patch<patches/simpler-
stackprint-deepinsight.s.patch>`
```

STM32F0

Patch: Move Terminal From USART1 to USART2

> Reason: To provide a byte of eleven consecutive bits, PA-4 to PA-15.
>
> - USART 1 is the default
> - Both USART1 and USART2 are enabled with SF1 which is also default

Board	**MCU**	**Comms Type**	**TX**	**RX**
STM32F0 Discovery	STM32F051R8	USART1	PA9	PA10
or				

Board	MCU	Comms Type	TX	RX
STM32F0 Discovery	STM32F051R8	USART2	PA2	PA3

- Applies to: mecrisp-stellaris-2.3.6/mecrisp-stellaris-source/stm32f051/terminal.s
- **`move terminal from usart1 to usart2 patch`**

Patch: Add RTS/CTS Handshaking To USART1

Reason, so the MCU UART1 can tell the PC terminal to stop sending data when required. The PC can also tell the MCU to stop sending data.

GPIO	Flow control	In/Out	ALT Function	Notes: PC is probably connected to a USB/3/3v serial dongle
PA12	USART1_RTS	OUT	AF1	output to PC (CTS) to STOP sending data when LOW
PA11	USART1_CTS	IN	AF1	input from PC (RTS), it’s CLEAR to send data when LOW

GPIO	Flow control	In/Out	ALT Function	Notes: PC is probably connected to a USB/3/3v serial dongle
PA9	USART1_TX	OUT	AF1	to PC RX
PA10	USART1_RX	IN	AF1	to PC TX

- Applies to: mecrisp-stellaris-2.4.2/mecrisp-stellaris-source/stm32f051/terminal.s but may apply to all STM32F0 Cortex M0 chips
- If you use this patch and don't connect hardware handshaking, you'll need to take PA11 LOW with a jumper to talk to the MCU
- **`DOWNLOAD: terminal.rts.cts.s.patch`**
- For info on hardware handshaking, click here

Rebuild Your Kernel

Compiling Mecrisp-Stellaris is easy.

- You'll need none-arm-eabi installed, which is available for FreeBSD. OpenBSD and Linux (at least).

Compile example for mecrisp-stellaris-x.x.x

- cd mecrisp-stellaris-x.x.x/mecrisp-stellaris-source
- ./assemble

- ./tidyup
- The new image is now available in mecrisp-stellaris-x.x.x/your-mcu/mecrisp-stellaris-your-mcu.bin (except Linux)

STM32F103 PATCH

For mecrisp-stellaris-2.5.3

Download

https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/tp-allpatch-stm32f103ra-2.5.3.patch

Features

Serial Port Connection is USART2

Baud Rate: 460800, 8,N,1

USART2	MCU Signal	3.3v/USB Dongle
PA1	RTS	CTS
PA2	TX	RX
PA3	RX	TX

- RED error warnings
- Register print Word enabled

```
.regs
lr: 000023E1
r0: 0000291C
r1: 00000000
r2: 0000291C
r3: 00000000
r4: 00005278
r5: 20004F3C
r6: 20004F8C
r7: 20000288
sp: 20000364
```

- No stair stepping, no need to add a \r in your terminal configuration
- Hardware handshaking, see table above
- uncluttered stack print

```
1  ok.
2  ok.
3  ok.
.s [3 ] 536891276 1 2 3  ok.
```

Patching

Directory Layout

To apply the patch.

```
.
├── mecrisp-stellaris-2.5.3
└── tp-allpatch-stm32f103ra-2.5.3.patch
```

Patch Command

```
patch < tp-allpatch-stm32f103ra-2.5.3.patch
```

Build Guide

1. Run patch command above

2. Cd to mecrisp-stellaris-2.5.3/mecrisp-stellaris-source/stm32f103-ra
3. Run gmake (freebsd) or make
4. Be sure to erase ALL the Flash on the chip FIRST.
5. Flash mecrisp-stellaris-2.5.3/mecrisp-stellaris-source/stm32f103-ra/mecrisp-stellaris-stm32f103.bin to your STM32F103 chip.

Porters Guide

Your chip isn’t listed in the suported hardware list, but you’d like to see if you can get it going, what to do ?

1) Find a close relative

Look through the suported hardware list and see if you can find a similar chip, one from the same family, remember even a image for a CortexM0 will run on a CortexM4 etc as they’re all upwards compatible.

2) Install the Image from a similar chip onto your chip/board

Using the recommended method just flash the Image youve decided is the closest fit.

Warning

Make sure your chip has enough Flash to accomodate the Image, but this is rarely a problem as most Mecrisp-Stellaris Images are less than 20KB and will run in a chip with 32KB Flash. If your chip has only 16KB Flash or less, chances are it won’t run, or run well. Even if it does run, you won’t have any room for application code or libraries etc.

Warning

I have observed that failure to fully erase the Flash memory prior to flashing Mecrisp-Stellaris often results in endless “Unhandled Interrupt 00000003” messages

3) Connect a USB/3/3v Dongle

Connect a serial usb/3.3 volt dongle to your PC. Run Picocom with the following command:

```
picocom -f h -b 115200 /dev/cuaU0 --imap lfcrlf,crcrlf --omap
delbs,crlf
```

Warning

change /dev/cuaU0 to suit your serial device, and if you don’t have permissions to use it, you may have to run this as root.

Double check the Terminal

Loopback test:

Connect the 3.3v TX and RX wires from your serial dongle together and type something in your terminal, what you type should be echoed back exactly.

Connect to chip/board

Look in the Serial Connections Table and connect the RX and TX wires from the dongle to the chip/board.

Note

remember TX –> RX and RX –> TX, terminal – chip/board

Now press the reset button on your board/chip and look for the Mecrisp-Stellaris Greeting…

If you don’t get it, try the other Serial Uarts on your chip, try other pins, perhaps it has a usb connector you can try plugging into your pc to try ?

Warning

I have transposed the TX and RX many times, so make sure that you have the right connections.

Garbled Text

You get garbled (unreadable) text when you press the chip/board reset button ?

Sounds like the wrong Baud Rate, but you’re *very close* to getting it working

4) It’s running!

Congrats, well done!

5) Change Memory Values

To suit your MCU variant

Testing Memory

See the Memstat utility.

Warning

Ram and Flash values reported by Memstat are totally seperate from values the Mecrisp-Stellaris Kernel uses. These values are preset and used when the Kernel is compiled. To configure the Kernel memory values for your MCU see Changing Kernel Flash and Ram values. Using these preset Kernel memory values, Mecrisp-Stellaris will print warnings when Flash or Ram reserves are exhausted.

Changing Kernel Flash and Ram values

Change the Memory config in mecrisp-stellaris-X.X.X/mecrisp-stellaris-source/your-mcu/your-mcu.s/ so the actual MCU memory will be available to Mecrisp-Stellaris to use.

```
@ Konstanten für die Größe des Ram-Speichers

.equ RamAnfang, 0x20000000 @ Start of RAM          Porting: Change this !
.equ RamEnde,   0x20002000 @ End   of RAM.   8 kb. Porting: Change this !

@ Konstanten für die Größe und Aufteilung des Flash-Speichers

.equ Kernschutzadresse,     0x00004000 @ Darunter wird niemals etwas geschrieben ! Mecrisp core never writes flash below this address.
.equ FlashDictionaryAnfang, 0x00004000 @ 16 kb für den Kern reserviert...           16 kb Flash reserved for core.
.equ FlashDictionaryEnde,   0x00010000 @ 48 kb Platz für das Flash-Dictionary       48 kb Flash available. Porting: Change this
.equ Backlinkgrenze,        RamAnfang  @ Ab dem Ram-Start.
```

Note

Mecrisp-Stellaris checks Ram and Flash used against the values in RamEnde and FlashDictionaryEnde and will print warnings when no more memory is available.

Memory Full Warning Examples

```
3 OF lmt01-3.power.on ENDOF   Ram full         <---- RAM FULL ERROR !!!

nvic_icer_clrena-tim6         Flash full       <---- FLASH FULL ERROR !!!
```

6) Changing MCU Name

```
Edit mecrisp-stellaris-X.X.X/mecrisp-stellaris-source/common/datastackandmacros.s line 91 and change the name:

 .macro welcome Meldung
  bl dotgaensefuesschen
     .byte 8f - 7f         @ Compute length of name field.
-      .ascii "Mecrisp-Stellaris RA X.X.X"
+      .ascii "Mecrisp-Stellaris RA X.X.X for mcu-xyz"
```

7) Building a new kernel

See rebuilding the kernel

Rename your mecrisp-stellaris-X.X.X.bin to mecrisp-stellaris-X.X.X-for-mcu-xyz.bin

Test New Kernel

- is the name right ?
- is the memory right ?

Checksum The Kernel

```
md5sum mecrisp-stellaris-X.X.X-for-mcu-xyz.bin
```

8) Write a README

A short for users, telling them what board/cpu this is for, how to connect, how to Flash and any other interesting details.

Email me a copy of the image, md5sum, your README and your name/email (whatever you want to release online) and I’ll put it on my doc site for users to download, and forward to you any emails I receive about it.

I’ll also email Matthias Koch who may add it to the official release, and even put your name on the famed list of contributors.

Flash Memory Operation

Mecrisp-Stellaris internally uses individual 2 byte writes (=opcode size!) and has buffer layers for ECC flash controllers with up to 16-bytes-at-once writes.

This generally means that MCU’s with Flash requiring *more than 16-bytes-at-once writes* cannot be supported.

Problem MCU’s

These MCU’s are problematic to port to Mecrisp-StellarisForth and therefore have no support.

Atmel SAM based chips

The Atmel series chip has a flash controller which supports only 256-bytes-at-once writes.

Mecrisp needs at least two separate writes for every definition, and every definition being at least 2*256 bytes large simply isn’t practical.

There is one contributed experimental target for atsame70, but it lacks “compiletoflash” mode, in other words Flash can’t be used, only Ram.

If you want to port a Atmel SAM based MCU, you’ll need to first check the datasheet on flash write granularity.

Examples Atmel Sam Boards

Adafruit Trinket M0

Arduino Due, Genuino

Cypress PSoC’s

Horribly bad documentation which makes finding out any details on initialisation very, very difficult.

Project: LMT01 Temperature Sensor

This project displays up to three LMT01 temperature sensor readings in degrees C/F, reading one sensor at a time. The design allows for one LMT01 sensor per GPIO pin, and more can be easily added.

Example Output

```
testall ( only LMT01 sensors 1 and 2 are fitted )

Sensor-1: 25.5C or 77.9F
Sensor-2: 25.4C or 77.7F
Sensor-3: LMT01 faulty or disconnected
Sensor-4: No such sensor !!  ok.
```

YouTube Video

https://youtu.be/UpIYeZxftcE

The LMT01 Temperature Sensor

The LMT01 provides a digital output in the form of a pulse count that is transmitted by a train of current pulses.

After the LMT01 is powered up, it transmits a very low current of 34 µA for less than 54 ms while performing a temperature to digital conversion. When the temperature-to-digital conversion is complete, the LMT01 transmits a pulse train that toggles from the low current of 34 µA to a high current level of 125 µA.

The total pulse count represents the temperature of the LMT01.

LMT01 Spec

The LMT01 outputs at minimum 1 pulse and a theoretical maximum of 4095 pulses. Each pulse has a weight of 0.0625°C.

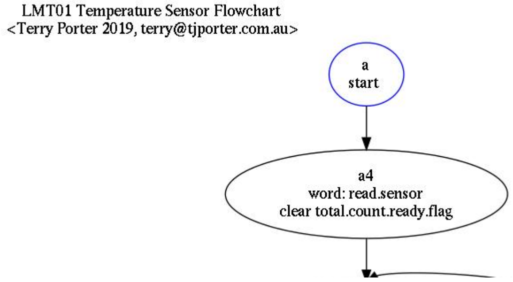
One pulse corresponds to a temperature less than -50°C while a pulse count of 4096 corresponds to a temperature greater than 200°C.

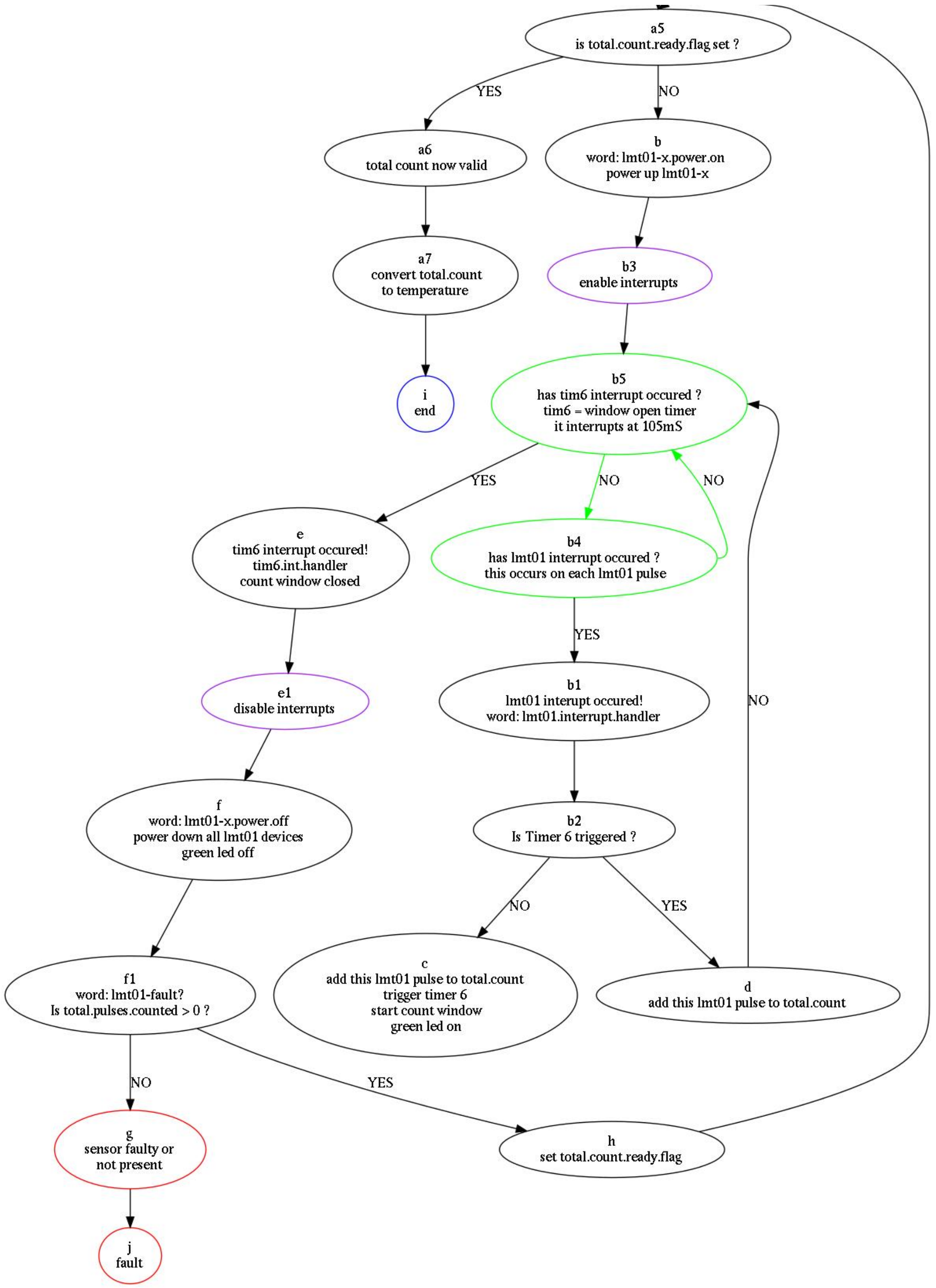
Warning

The LMT01 is only ensured to operate up to 150°C. Exceeding this temperature by more than 5°C may damage the device. The accuracy of the device degrades as well when 150°C is exceeded.

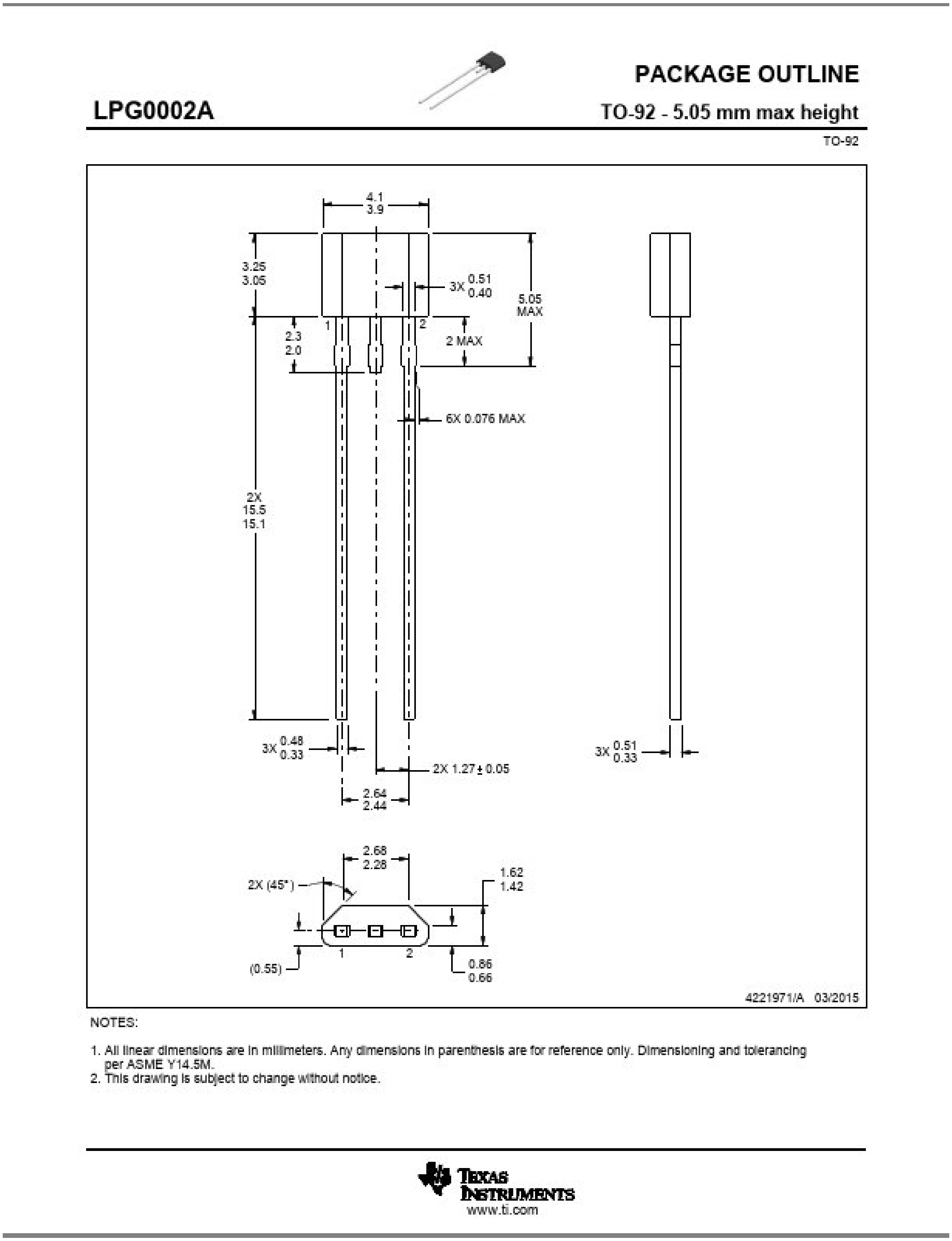
Software Flowchart

This is the basic design of the Forth software to read multiple LMT01 two wire temperature sensors. My next modification will be to count the pulses from the temperature sensors in a counter timer so the MCU doesn’t need to do it (currently handled by interrupts), freeing up cpu cycles for more important tasks.


LMT01 Temperature Sensor Flowchart
<Terry Porter 2019, terry@tjporter.com.au>
a
start
a4
word: read.sensor
clear total.count.ready.flag


a5
is total.count.ready.flag set ?
YES
NO
a6
total count now valid
b
word: lmt01-x.power.on
power up lmt01-x
a7
convert total.count
to temperature
b3
enable interrupts
i
end
b5
has tim6 interrupt occured ?
tim6 = window open timer
it interrupts at 105mS
YES
NO
NO
e
tim6 interrupt occured!
tim6.int.handler
count window closed
b4
has lmt01 interrupt occured ?
this occurs on each lmt01 pulse
YES
e1
disable interrupts
b1
lmt01 interupt occured!
word: lmt01.interrupt.handler
NO
f
word: lmt01-x.power.off
power down all lmt01 devices
green led off
b2
Is Timer 6 triggered ?
NO
YES
f1
word: lmt01-fault?
Is total.pulses.counted > 0 ?
c
add this lmt01 pulse to total.count
trigger timer 6
start count window
green led on
d
add this lmt01 pulse to total.count
NO
YES
g
sensor faulty or
not present
h
set total.count.ready.flag
j
fault

LMT01 TO-92 Package



PACKAGE OUTLINE

LPG0002A

TO-92 - 5.05 mm max height

TO-92

4.1
3.9
3.25
3.05
3X 0.51
0.40
5.05
MAX
1
2
2.3
2.0
2 MAX
6X 0.076 MAX
2X
15.5
15.1
3X 0.48
0.33
2X 1.27 ± 0.05
2.64
2.44
3X 0.51
0.33
2.68
2.28
2X (45°)
1.62
1.42
(0.55)
0.86
0.66

4221971/A 03/2015

NOTES:

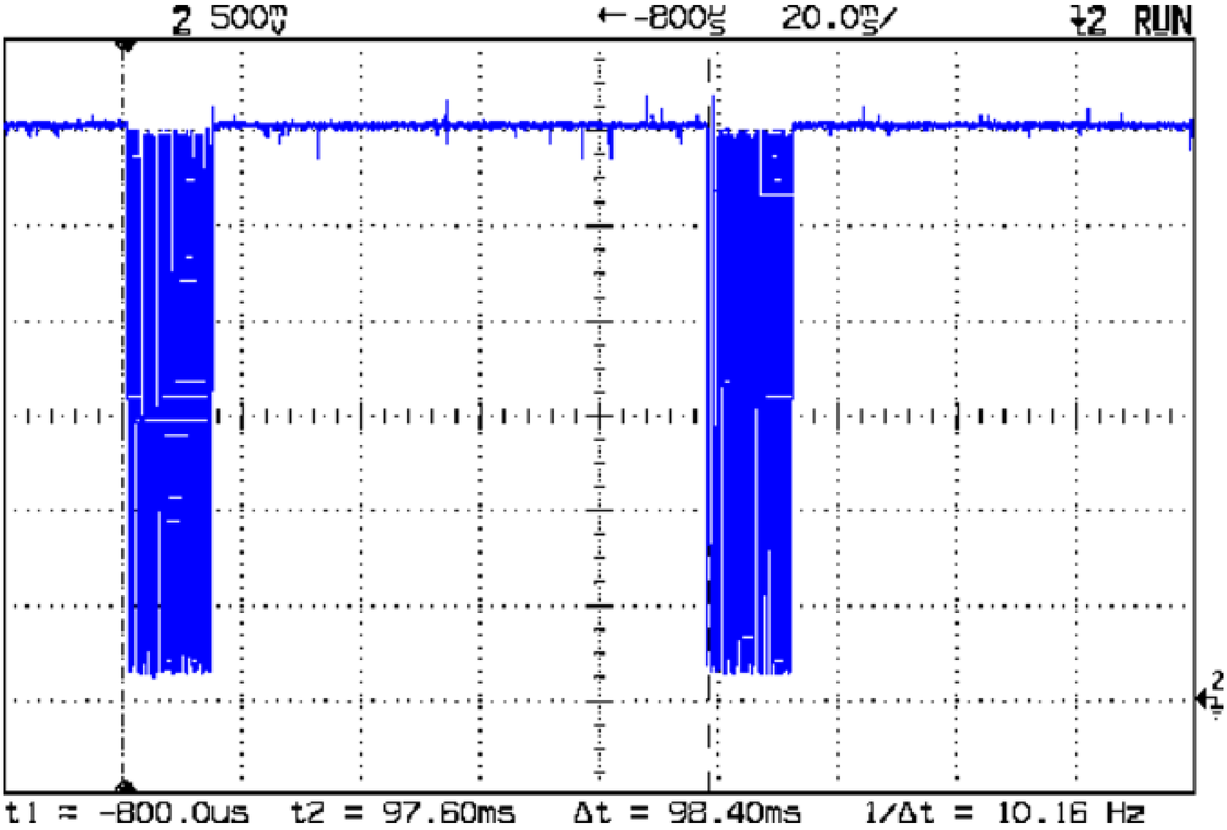
1. All linear dimensions are in millimeters. Any dimensions in parenthesis are for reference only. Dimensioning and tolerancing per ASME Y14.5M.
2. This drawing is subject to change without notice.

TEXAS INSTRUMENTS
www.ti.com

LMT01 Waveforms

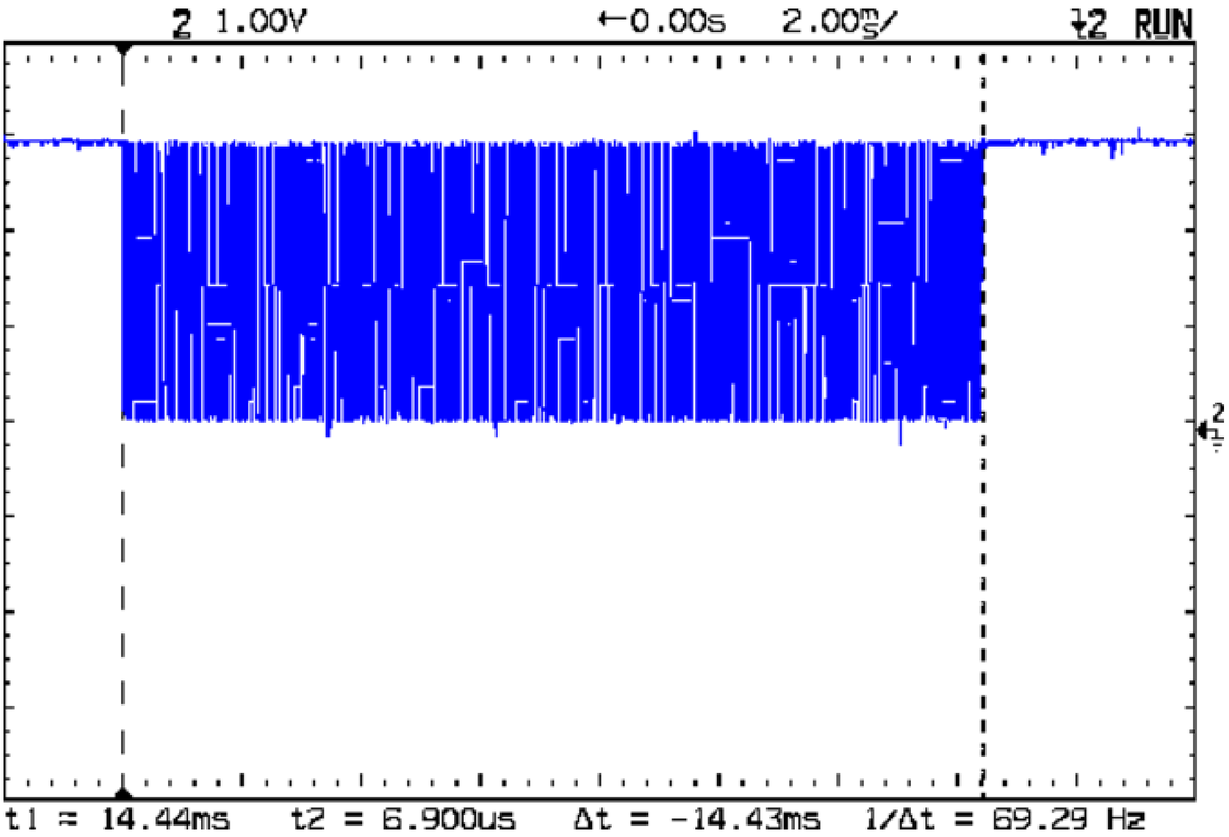
Free Running Pulse Block Timing

Taken at the Collector of Q1


2 500mV
←-800µs
20.0ms/
↓2 RUN
2
t1 = -800.0us
t2 = 97.60ms
Δt = 98.40ms
1/Δt = 10.16 Hz

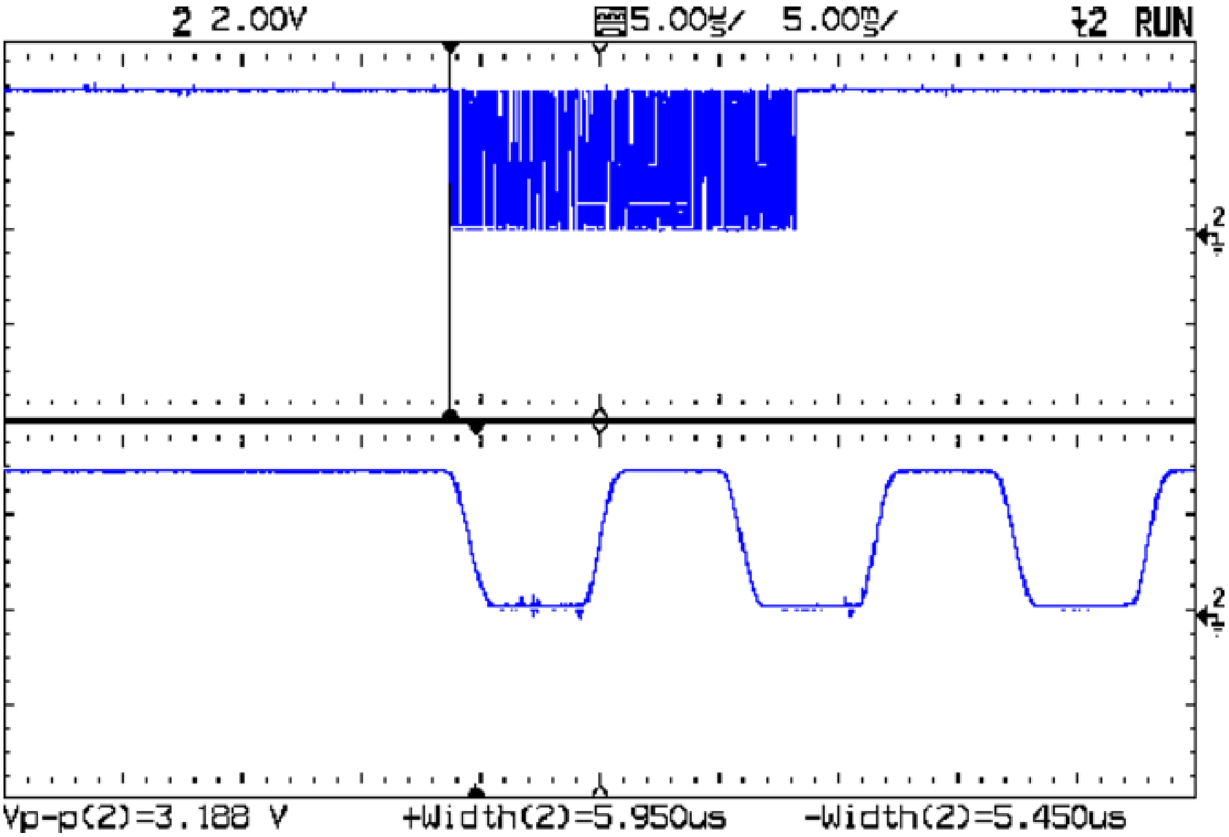
Pulse Block Width

Taken at the Collector of Q1



Individual Pulse Timing

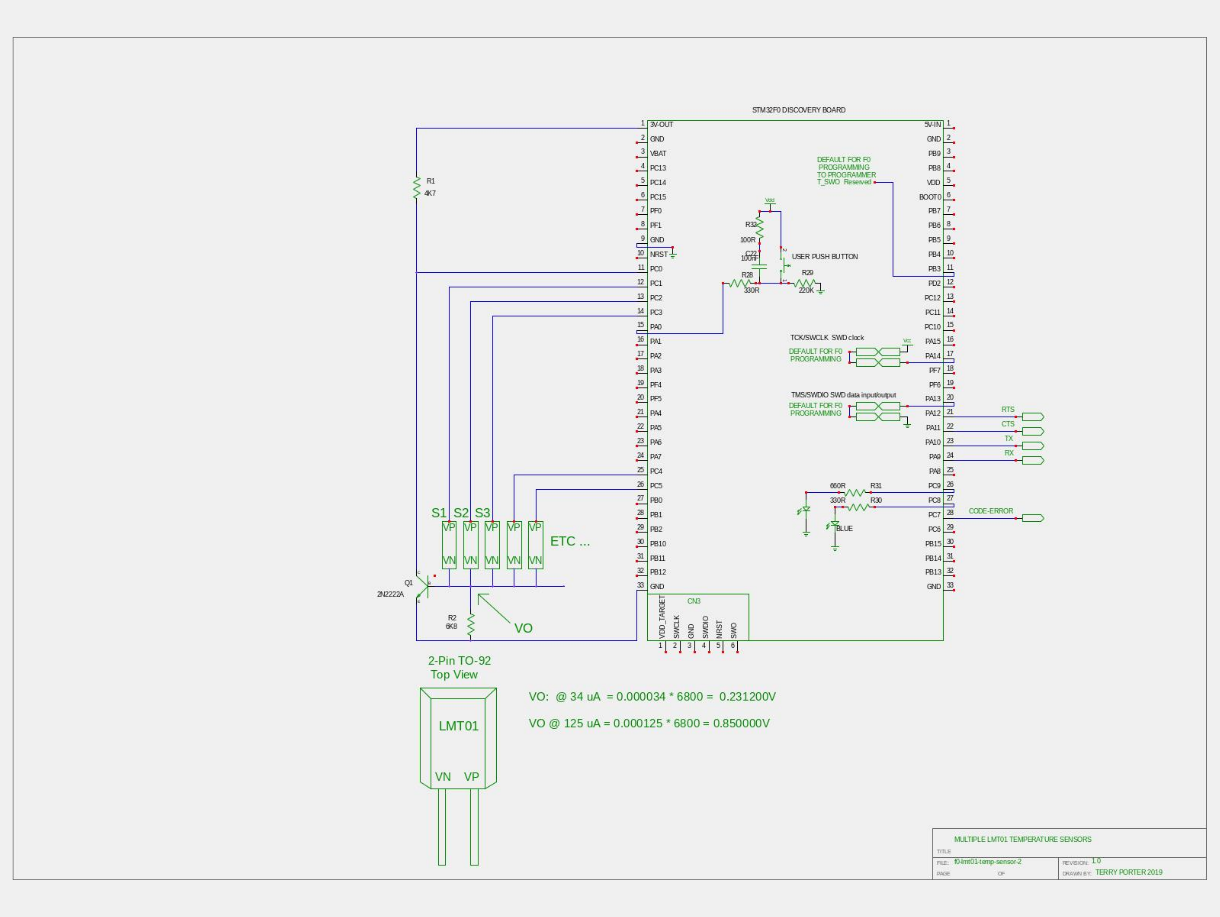
Taken at the Collector of Q1


2 2.00V
5.00µs/ 5.00ms/
↓2 RUN
2
2
Vp-p(2)=3.188 V
+Width(2)=5.950us
-Width(2)=5.450us

The schematic

Note

all parts inside the STM32F0 Discovery Board rectangle are actually built into it by ST, they aren’t parts I’ve added for this project


STM32F0 DISCOVERY BOARD
1 3V-OUT
2 GND
3 VBAT
4 PC13
5 PC14
6 PC15
7 PF0
8 PF1
9 GND
10 NRST
11 PC0
12 PC1
13 PC2
14 PC3
15 PA0
16 PA1
17 PA2
18 PA3
19 PF4
20 PF5
21 PA4
22 PA5
23 PA6
24 PA7
25 PC4
26 PC5
27 PB0
28 PB1
29 PB2
30 PB10
31 PB11
32 PB12
33 GND
5V-IN 1
GND 2
PB9 3
PB8 4
VDD 5
BOOT0 6
PB7 7
PB6 8
PB5 9
PB4 10
PB3 11
PD2 12
PC12 13
PC11 14
PC10 15
PA15 16
PA14 17
PF7 18
PF6 19
PA13 20
PA12 21
PA11 22
PA10 23
PA9 24
PA8 25
PC9 26
PC8 27
PC7 28
PC6 29
PB15 30
PB14 31
PB13 32
GND 33
R1
4K7
DEFAULT FOR F0 PROGRAMMING TO PROGRAMMER T_SWO Reserved
Vdd
R32
100R
C22
100nF
USER PUSH BUTTON
R28
330R
R29
220K
TCK/SWCLK SWD clock
DEFAULT FOR F0 PROGRAMMING
Vcc
TMS/SWDIO SWD data input/output
DEFAULT FOR F0 PROGRAMMING
RTS
CTS
TX
RX
660R
R31
330R
R30
BLUE
CODE-ERROR
S1 S2 S3
VP
VN
ETC ...
Q1
2N2222A
R2
6K8
VO
CN3
VDD_TARGET
SWCLK
GND
SWDIO
NRST
SWO
1 2 3 4 5 6
2-Pin TO-92
Top View
LMT01
VN VP
VO: @ 34 uA = 0.000034 * 6800 = 0.231200V
VO @ 125 uA = 0.000125 * 6800 = 0.850000V
MULTIPLE LMT01 TEMPERATURE SENSORS
TITLE
FILE: f0-lmt01-temp-sensor-2
PAGE OF
REVISION: 1.0
DRAWN BY: TERRY PORTER 2019

LMT01 Software

This is actually one file **`f0-lmt01-temp-sensor-2.fs`** but I've broken it up into three parts for clarity. If you copy and paste these files, just load part-1 first, part-2 second, and part-3 third.

Part-2

This is the 'main' program which counts the temperature sensor pulses, scales and prints the results.

```
: lmt01-fault? ( -- ) raw.temperature @ 0 = ;        \ If a
sensor has a zero count, it's faulty
: pretty.print ( -- )  0 <# # 46 hold #s #> type  ;  \ Insert
one decimal point into final value
```

```
: degrees.c ( raw -- C )  \ Print Degrees C
  pretty.print ." C "
;

: degrees.f ( raw -- F )  \ Convert Degrees C to Degrees F
(formula: °C  x  9/5 + 32 = °F
   90 5 / * 3200 + 10 /
   pretty.print ." F "
;

: degrees ( -- )
     lmt01-fault? if ." LMT01 faulty or disconnected "
        else
        raw.temperature @                      \ fetch total
number of pulses counted
        10000 4096 */ 256 * 500000 - 1000 /    \ Temp (C) =
((count/4096) *256) -50 scale raw temp into degrees x 10
        dup degrees.c ." or " degrees.f
     then
;

: temperature? ( sensor -- )             \ one temperature?
measurement
  0 lmt01-x.count !                      \ lmt01-x.count
increments once per lmt01-x pulse
  0 raw.temperature !                    \ raw.temperature is
lmt01-x.count total count
  0 ready.flag !                         \ ready.flag is set
when raw.temperature value is stable and ready to be used
     case                                \ power up selected
sensor
        1 OF lmt01-1.power.on ENDOF
        2 OF lmt01-2.power.on ENDOF
        3 OF lmt01-3.power.on ENDOF
        ." No such sensor !! " drop exit         \ sensor not
implimented fall thru warning
     endcase
  green-on                               \ mark start of
count window
  tim6_cr1_cen                           \ start timer6 (end
of timer kills the power to the lmt01)
     begin                               \ loop until
raw.temperature is stable
        pause
        ready.flag @ 1 =
     until
  degrees cr
;

: t temperature? ;

: testall cr ( -- ) \ simple test of sensors
  ." Sensor-1: " 1 t
  ." Sensor-2: " 2 t
  ." Sensor-3: " 3 t
  ." Sensor-4: " 4 t
;
```

Part-1

Contains all the low level routines, variables and constants

```
compiletoram

0 variable lmt01-x.count
0 variable raw.temperature
0 variable ready.flag

\ RCC Words
: rcc_apb2enr_syscfgen-set ( -- )  %1  0  lshift  rcc_apb2enr
bis! ;   \ rcc_apb2enr_syscfgen syscfg clock enable
: rcc_apb1enr_tim6en ( -- ) %1 4 lshift rcc_apb1enr bis! ;
\ rcc_apb1enr_tim6en tim6 clock enable

: init.gpio ( -- )
  \ One Port pull up for all sensors
  rcc_apb2enr_syscfgen-set   \ REQUIRED!! for using GPIO's as
interrupts!
  pull-up ( %xx -- ) 0 lshift gpioc_pupdr bis!  \
gpioc_pupdr_pupdr0 PC0
;

\ LMT01 related Words
: syscfg_exticr1_exti0 ( -- ) %0010  0  lshift
syscfg_exticr1 bis! ;  \ pc-0 is interrupt source  x010:
pc[0] pin
: exti_imr_mr0-set ( -- )  %1  0  lshift  exti_imr bis! ;
\ exti_imr_mr0 set interrupt mask on line 0
```

```
: exti_pr_pr0 ( -- ) %1 0 lshift exti_pr bis! ;
\ exti_pr_pr0    pending bit 0
: exti_ftsr_tr0-set ( -- )  %1  0  lshift  exti_ftsr bis! ; \
exti_ftsr_tr0 falling trigger event configuration of  line 0
: exti_pr_pr0-set ( -- )  %1  0  lshift exti_pr  bis! ;
\ exti_pr_pr0 interrupt pending flag for line 0
: nvic_iser_setena-exti0_1 ( -- )  %100000 0  lshift
nvic_iser  bis! ; \ nvic_iser_setena-exti0_1
: nvic_icer_clrena-exti0_1 ( -- )  %100000 0  lshift
nvic_icer  bis! ; \ nvic_icer_clrena-exti0_1

: lmt01-1.power.on  ( -- )
  output  ( %xx -- ) 2 lshift gpioc_moder bis!    \
gpioc_moder_moder1  PC1
  %1 1 lshift gpioc_bsrr bis!                     \
gpioc_bsrr_bs1; output high
;

: lmt01-2.power.on  ( -- )
  output ( %XX -- ) 4 lshift GPIOC_MODER bis!     \
GPIOC_MODER_MODER2  PC2
  %1 2 lshift gpioc_bsrr bis! ;                   \
gpioc_bsrr_bs2; output high


: lmt01-3.power.on  ( -- )
  output ( %XX -- ) 6 lshift gpioc_moder bic!     \
gpioc_moder_moder3  PC3
  %1 3 lshift gpioc_bsrr bis! ;                   \
gpioc_bsrr_bs3; output high


: lmt01-1.power.off %11 ( -- ) 2 lshift gpioc_moder bic! ; \
gpioc_moder_moder1  PC1
: lmt01-2.power.off %11 ( -- ) 4 lshift gpioc_moder bic! ; \
gpioc_moder_moder2  PC2
: lmt01-3.power.off %11 ( -- ) 6 lshift gpioc_moder bic! ; \
gpioc_moder_moder3  PC3

: lmt01-x.power.off ( -- ) \ turn off all lmt01's by
switching the port from output to input mode
  lmt01-1.power.off
  lmt01-2.power.off
  lmt01-3.power.off
;
```

```
\ Tim6 related Words
48000000 constant fCK

: nvic_iser_setena-tim6 ( -- ) 1 17 lshift nvic_iser bis! ;
\ nvic_iser_setena   tim6
: nvic_icer_clrena-tim6 ( -- ) 1 17 lshift nvic_icer bis! ;
\ nvic_icer_clrena  tim6
: nvic_icpr_clrpend-tim6 ( -- ) 1 17 lshift nvic_icpr bis! ;
\ nvic_icpr_clrpend  tim6
: tim6_cnt@ ( -- ) tim6_cnt h@ ;  \ tim6_cnt_cnt   16 bit
counter value
: tim6_cr1_opm ( -- ) %1 3 lshift tim6_cr1 hbis! ;      \
tim6_cr1_opm one-pulse mode
: tim6_cr1_cen ( -- ) %1 0 lshift tim6_cr1 hbis! ;      \
tim6_cr1_cen    counter enable
: tim6_cr1_cdis ( -- ) %1 0 lshift tim6_cr1 hbic! ;     \
tim6_cr1_cen    counter disable
: tim6_dier_uie ( -- ) %1 0 lshift tim6_dier hbis! ;    \
tim6_dier_uie        update interrupt enable
: tim6_sr_uif ( -- ) %1 0 lshift tim6_sr hbic! ;        \
tim6_sr_uif  clear tim6 int pending flag
: tim6_egr_ug ( -- ) %1 0 lshift tim6_egr hbis! ;       \
tim6_egr_ug  update generation

: preset.psc.no.int ( -- ) \ preset Prescaler but don't cause
interrupt. PSC does not get 'active' until the second update
event
  tim6_arr h!              \ preset arr
  tim6_egr_ug              \ update generation: (causes a
interrupt when run) but loads ARR into shaddow register!
  tim6_sr_uif              \ clear tim6 int pending flag
after running tim6_egr_ug
  tim6_dier_uie            \ update interrupt enable
;

: tim6.int.handler ( -- )
  nvic_icpr_clrpend-tim6              \ clear pending iser
for tim6
  tim6_sr_uif                         \ clear tim6 int
pending flag
  tim6_cr1_cdis                       \ counter disable
  nvic_icer_clrena-tim6               \ disable the tim6
interrupt or dropping power to the LMT01 will re-trigger it
  green-off                           \ mark finish of count
```

```
window
  lmt01-x.power.off
  lmt01-x.count @ raw.temperature !   \ raw.temperature now
contains valid lmt01 count
  1 ready.flag !                      \ it's safe to read
raw.temperature now
  nvic_iser_setena-tim6               \ re-enable tim6 iser
  nvic_iser_setena-exti0_1            \ re enable exti0_1
iser ( won't work without this )
;

: init.tim6 ( -- )
  rcc_apb1enr_tim6en      \ enable tim6 in rcc
  tim6_cr1_opm            \ one pulse mode high = Counter
stops counting at the next update event (clearing the bit
CEN)
  47952 tim6_psc h!       \ preset prescaler for 1mS (47 for
1 uS)
  104 preset.psc.no.int   \ preset PSC --> clear int pending
flag, enable interrupt
  ['] tim6.int.handler irq-tim6_dac !
  nvic_iser_setena-tim6   \ enable iser
;

: lmt01.interrupt.handler ( -- )
  exti_pr_pr0                         \ clear pending bit 0
  lmt01-x.count @ 1+ lmt01-x.count !  \ increment temperature
count by one
  exti_pr_pr0-set                     \ clear external
interrupt pending register (exti_pr)
;

: init.lmt01 ( -- )
  syscfg_exticr1_exti0                         \ negative
going pulses from lmt01 trigger pc0 interrupt
  exti_imr_mr0-set                             \ exti0 set
interrupt mask on line 0
  exti_ftsr_tr0-set                            \ falling
trigger on line 0
  ['] lmt01.interrupt.handler irq-exti0_1 !    \ tie the
interrupt handler to the interrupt
  nvic_iser_setena-exti0_1                     \ enable
exti0_1 interrupt
;
```

Part-3

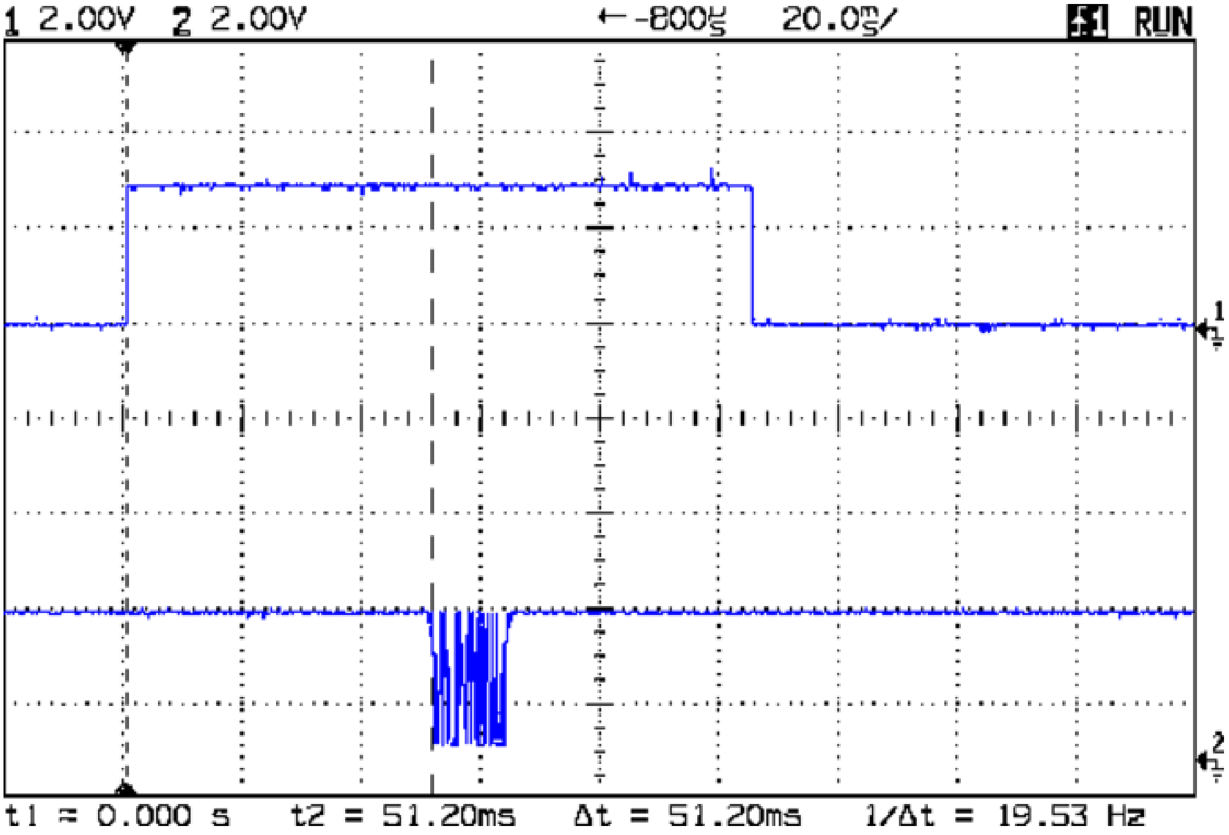
This last part adds initialization and multitasking

```
: main ( -- )                \ executed about every 5ms
@48mHz
 pause
;

: init-general ( -- )        \ executed once at boot-up
  f0-disco-48mhz             \ switch MCU clock from 8 to 48
MHz
  init.gpio                  \ Initialisations
  init.tim6
  init.lmt01
;

task: maintask         \ essential multitasking Word
: main& ( -- )          \ multitasking.fs Word
  maintask activate    \ multitasking.fs Words
  begin main again     \ this is the only task, and it loops
forever (about every 5uS @ 48Mhz clock) unless blocked
;

: init ( -- )        \ essential multitasking Word
  init-general      \ main user application inits
  multitask         \ multitasking.fs Word
  main&             \ multitasking.fs Word
;

init                 \ this init for non flash use, i.e.
after uploading this program
```

Waveforms

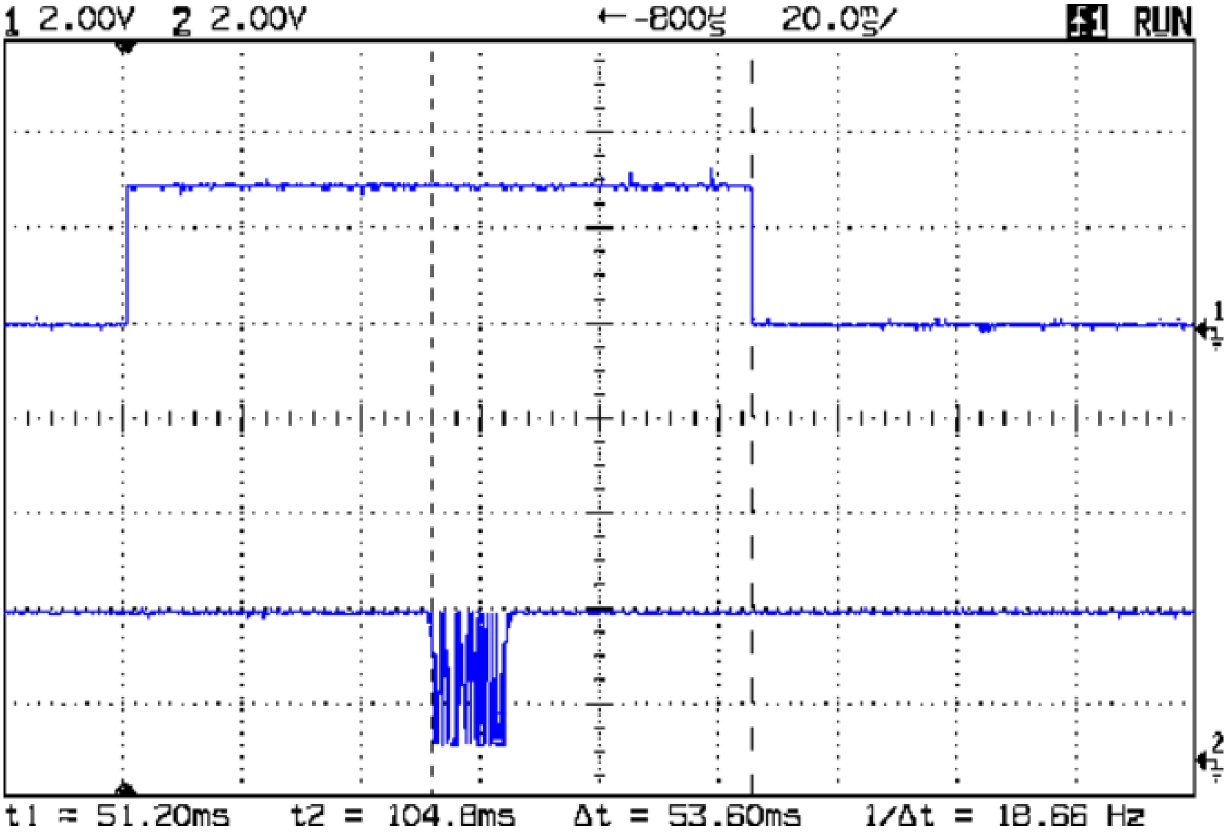
Window Start to Pulse Block Start Timing

- Window and Power are the same thing displayed on CH-1
- Waveform 1: PC-1
- Waveform 2: Collector Q1



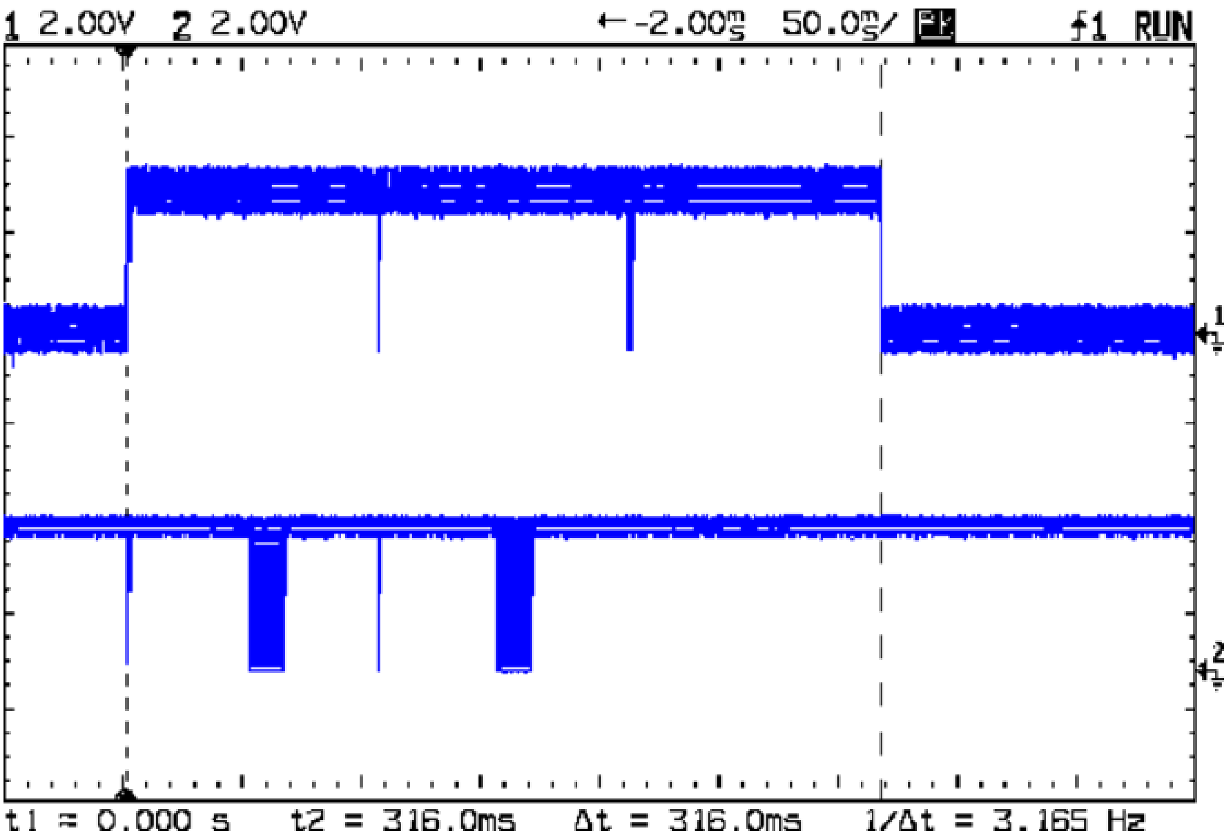
Pulse Block Start to Window Close Timing

- Waveform 1: PC-1
- Waveform 2: Collector Q1



Testall

- Waveform 1: PC-1
- Waveform 2: Collector Q1


1 2.00V 2 2.00V
←-2.00ms 50.0ms/
f1 RUN
1
2
t1 = 0.000 s t2 = 316.0ms Δt = 316.0ms 1/Δt = 3.165 Hz

Downloads

f0-lmt01-temp-sensor-2.fs

memmap.fs

bitfields.fs

template.xml

screen-f0-lmt01-temp-sens~2-cuaU0-498600-ctsrts.sh

Kernel be337881895e64e60016a3514a0dea8e.bin

be337881895e64e60016a3514a0dea8e.words.txt

be337881895e64e60016a3514a0dea8e.README.txt

lmt01.pdf

Note

The Kernel binary filename is a MD5SUM of the file itself.

```
md5 be337881895e64e60016a3514a0dea8e.bin
MD5 (be337881895e64e60016a3514a0dea8e.bin) =
be337881895e64e60016a3514a0dea8e
```

A Blue Pill LMT01 Temperature Sensor

This is a rewrite of the previous LMT01 temperature sensor project. It now uses fixed point arithmetic to calculate the temperature to one decimal place and has some extra features such as USB/USART2 terminal access, and a MCU “Device Electronic Signature” reader etc.

It contains a ready to run binary for a 64KB Flash version of the STM32F103 Cortex-M3 MCU as used in the Blue Pill that boots from USB at power up.

Connect the few components shown in the schematic below to read the temperature. It runs fine without them but should complain about a missing sensor if you try to read the temperature.

USB

This uses the Mecrisp-Stellaris USB driver for STM32F103 by Jean-Claude Wippler which is based on the Coreforth USB driver by Eckhart Köppen.

Note

From Viktor Dvořák. USB needs PC side idVendor = 0x0483 idProduct = 0x7540 driver which is STM32 Virtual COM port driver desribed in en.DM00478229.pdf. The driver is en.stsw-stm32102.zip (24755 kB) is available directly from st.com

- Connect the “Blue Pill” USB port to PC and look for “<Mecrisp STM32F10x Forth Serial Port>” device in your system
- Connect a PC serial terminal to the USB device above, you can choose any Baud Rate.

Note

The USB terminal can switch to USART2 by entering “-usb”. Speed is 460800 Baud.

USART2

Connect a serial terminal to read the temperature.

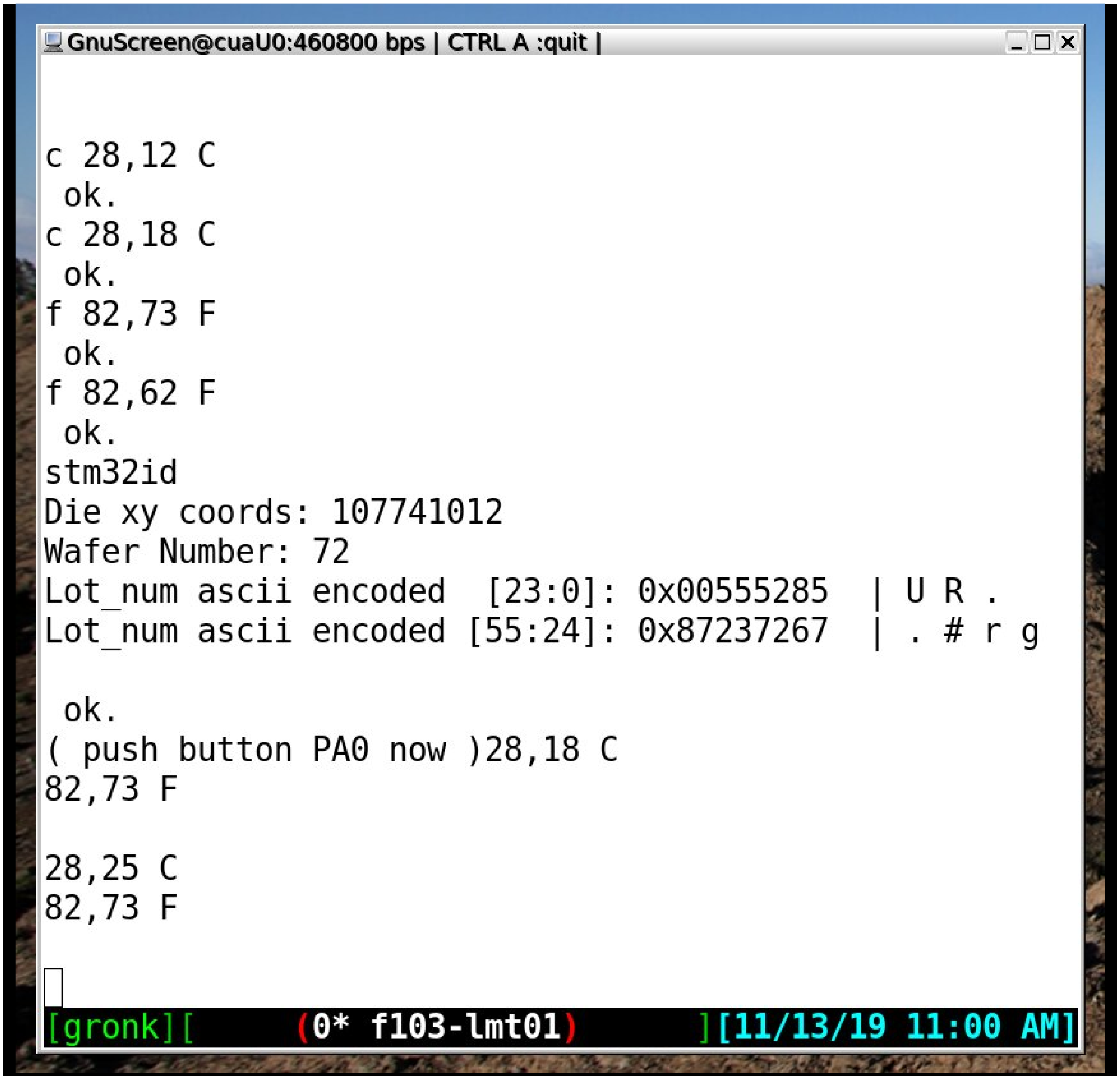
USART2	MCU Signal	3.3v/USB Dongle
PA1	RTS	CTS
PA2	TX	RX
PA3	RX	TX

Note

The USART2 terminal can switch to USB by entering “+usb”

Screenshot

- read the temperature by entering “C” or “F” in the terminal.
- check the mcu signature by entering “stm32id”
- examine the contents of the RCC CR register by entering “rcc_cr.”
- read the remperature by pressing the PA0 pushbutton



```
c 28,12 C
 ok.
c 28,18 C
 ok.
f 82,73 F
 ok.
f 82,62 F
 ok.
stm32id
Die xy coords: 107741012
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00555285  | U R .
Lot_num ascii encoded [55:24]: 0x87237267  | . # r g

 ok.
( push button PA0 now )28,18 C
82,73 F

28,25 C
82,73 F


[gronk][     (0* f103-lmt01)      ][11/13/19 11:00 AM]
```

Instructions

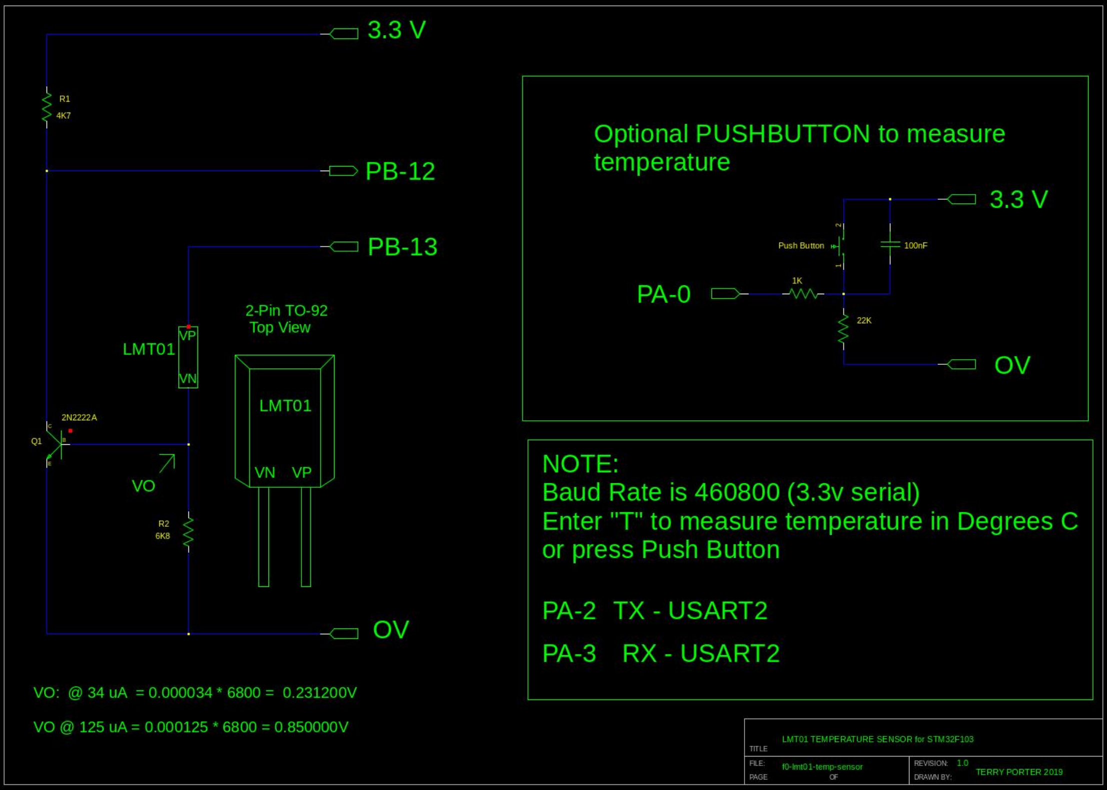
- Flash the ae9236b921613c07b1967f21ff20ee97.bin image in this tarball to your STM32F103RBT6 (128KB Flash version) board.
- Download it here: https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/ae9236b921613c07b1967f21ff20ee97.tar.gz/download
- Connect up the few components as shown in the schematic below and power up the board.
- The temperature may be read on the terminal by pressing the pushbutton, or entering “T” in the terminal.

- The Board runs at 72 MHz using the on board 8MHz crystal.

Schematic

Note

Press “C” or “F” for temperature reading, not “C” as shown below on this earlier version schematic.


3.3 V
R1
4K7
PB-12
PB-13
2-Pin TO-92
Top View
LMT01
VP
VN
LMT01
VN VP
2N2222A
Q1
C
B
E
VO
R2
6K8
OV
Optional PUSHBUTTON to measure temperature
3.3 V
Push Button
2
1
100nF
PA-0
1K
22K
OV
TITLE
LMT01 TEMPERATURE SENSOR for STM32F103
FILE:
f0-lmt01-temp-sensor
PAGE
OF
REVISION:
1.0
DRAWN BY:
TERRY PORTER 2019

NOTE:
Baud Rate is 460800 (3.3v serial)
Enter "T" to measure temperature in Degrees C
or press Push Button

PA-2 TX - USART2

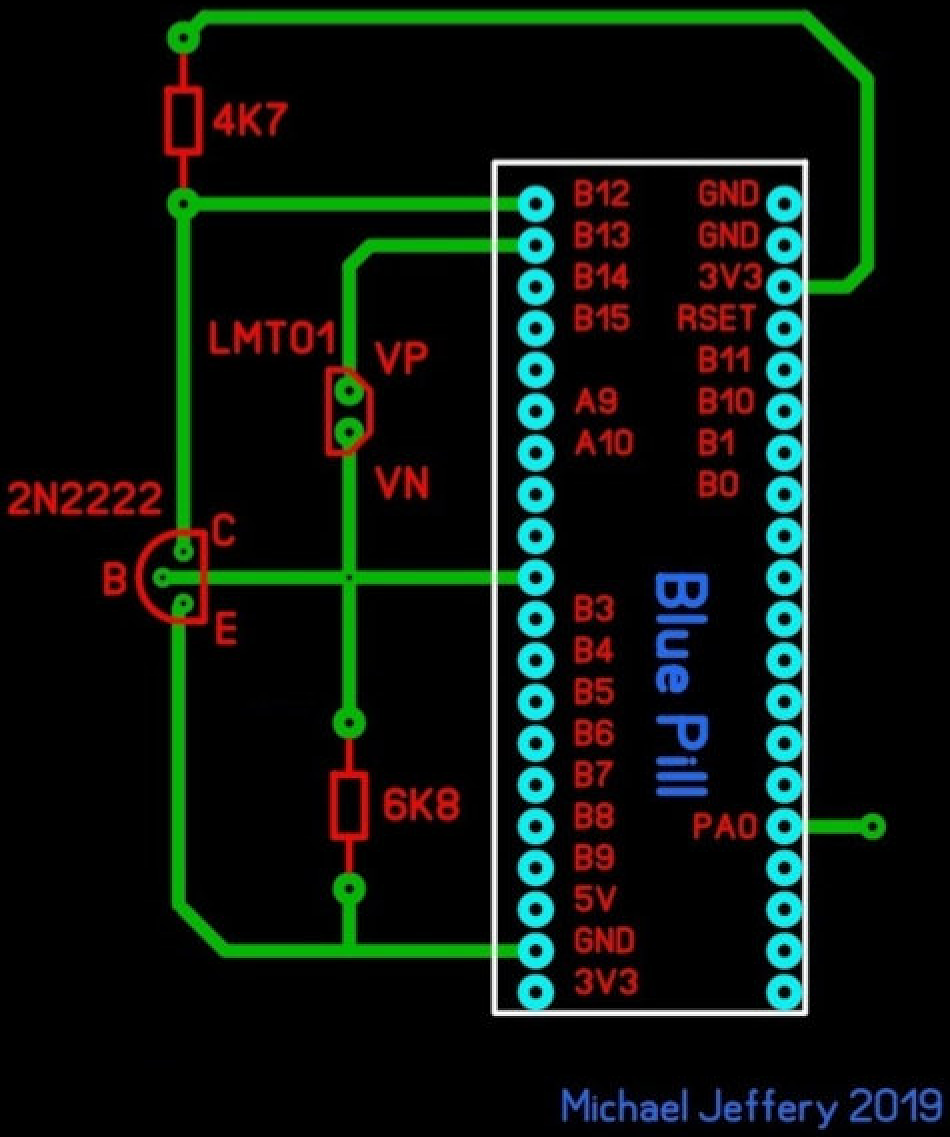
PA-3 RX - USART2

VO: @ 34 uA = 0.000034 * 6800 = 0.231200V

VO @ 125 uA = 0.000125 * 6800 = 0.850000V

Blue Pill Layout

Layout contributed by Michael Jeffery.


4K7
LMT01
VP
VN
2N2222
C
B
E
6K8
B12
B13
B14
B15
A9
A10
B3
B4
B5
B6
B7
B8
B9
5V
GND
3V3
GND
GND
3V3
RSET
B11
B10
B1
B0
PA0
Blue Pill
Michael Jeffery 2019

Tarball Contents

ae9236b921613c07b1967f21ff20ee97.tar.gz

```
.
├── ae9236b921613c07b1967f21ff20ee97.DELETE.sh
├── ae9236b921613c07b1967f21ff20ee97.README.md
├── ae9236b921613c07b1967f21ff20ee97.bin
├── ae9236b921613c07b1967f21ff20ee97.flashme
├── ae9236b921613c07b1967f21ff20ee97.words4.txt
├── f103-lmt01.jpeg
├── image.bin
└── lmt01-blue-pill-layout-1.jpg
```

Source Code

This source code by itself won’t run the temperature sensor, however that’s no problem as all the support Words required are installed in the downloadable ready to run image above. Nothing else is required, just enter “C” or “F” in the terminal or press the push button to read the temperature.

Source Code

```
\ Program Name: f103-64KB-USB-LMT01.fs
\ Date: Sun 10 Nov 2019 14:53:35 AEDT
\ Copyright 2019 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV2
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F103, Board: Olimex P103
\ Clock: 72MHz
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\ Requires preloaded support program(s):
\
\ This Program : LMT01 temperature reader with USB for a blue pill with 64KB flash
\
\ ---------------------------------------------------------------------------\
\ LMT01: Output Transfer Function: Temp (C) = ((count/4096) *256) -50
\    Temperature (°C)    Number of pulses
\         -49.9375      1
\          -49.875      2
\             -40      160
\             -20      480
\                0      800
\               30     1280
\               50     1600
\              100     2400
\              150     3200
\
\ ---------------------------------------------------------------------------\
\  GPIO Description
\  ---- -----------
\  PA0   Wake Up PB, normally low
\  PC12  LED ( open drain to led cathode) irq$ = 6
\  PB12  Count input                      irq# = 40
\  PB13  LMT01-1 Activate
\ ----------------------------------------------------------------------------\
\ compiletoram
compiletoflash

\ Variables
 0 variable count.lmt01


 \ GPIO Words
 : GPIOB_CRH_MODE13   ( %XX -- ) 20 lshift GPIOB_CRH bis! ;          \ GPIOB_CRH_MODE13 Port n.13 mode bits
 : GPIOB_CRH_CNF13   ( %XX -- ) 22 lshift GPIOB_CRH bic! ;           \ GPIOB_CRH_CNF13 Port n.13 configuration  bits
 : GPIOB_BSRR_BS13   %1 13 lshift GPIOB_BSRR bis! ; \ GPIOB_BSRR_BS13 Set bit 13
 : GPIOB_BSRR_BR13   %1 29 lshift GPIOB_BSRR bis! ; \ GPIOB_BSRR_BR13 Reset bit 13
 : GPIOA_CRL_CNF0   ( %XX -- ) 2 lshift GPIOA_CRL bis! ; \ GPIOA_CRL_CNF0 Port n.0 configuration  bits
 : GPIOA_BRR_BR0   %1 0 lshift GPIOA_BRR bis! ; \ GPIOA_BRR_BR0 Reset bit 0

 \ Interrupt Words for LMT01 on PB12, falling trigger. Interrupt #40
 : AFIO_EXTICR4_EXTI12   ( %XXXX -- ) 0 lshift AFIO_EXTICR4 bis! ;   \ AFIO_EXTICR4_EXTI12 EXTI12 configuration
 : line.12.unmask   %1 12 lshift EXTI_IMR bis! ; \ EXTI_IMR_MR12 Interrupt Mask on line 12
 : line.12.mask   %1 12 lshift EXTI_IMR bic! ; \ EXTI_IMR_MR12 Interrupt Mask on line 12
 : EXTI_RTSR_TR12   %1 12 lshift EXTI_RTSR bis! ; \ EXTI_RTSR_TR12 Rising trigger line 12
 : EXTI_FTSR_TR12   %1 12 lshift EXTI_FTSR bis! ; \ EXTI_FTSR_TR12 Falling trigger line 12
 : EXTI_PR_PR12   %1 12 lshift EXTI_PR bis! ; \ EXTI_PR_PR12 Pending bit 12
 : NVIC_ISER1_SETENA   ( %XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX -- ) 0 lshift NVIC_ISER1 bis! ;  \ NVIC_ISER1_SETENA    SETENA

 \ Interrupt for PB on PA0 falling edge as trigger. Interrupt #6
 : AFIO_EXTICR1_EXTI0   ( %XXXX -- ) 0 lshift AFIO_EXTICR1 bis! ;    \ EXTI0 configuration
 : line.0.unmask  %1 0 lshift EXTI_IMR bis! ; \ EXTI_IMR_MR0 Interrupt Mask on line 0
 : line.0.mask    %1 0 lshift EXTI_IMR bic! ;        \ EXTI_IMR_MR0 Interrupt Mask on line 0
 : EXTI_RTSR_TR0   %1 0 lshift EXTI_RTSR bis! ; \ EXTI_RTSR_TR0 Rising trigger line 0
 : EXTI_PR_PR0   %1 0 lshift EXTI_PR bis! ; \ EXTI_PR_PR0 Pending bit 0
 : NVIC_ISER0_SETENA   ( %XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX -- ) 0 lshift NVIC_ISER0 bis! ;  \ NVIC_ISER0_SETENA    SETENA

 \ General Words
 : >s31.32 ( u -- s31.32 ) 0 swap ; \ convert integer to a Fixed Point s31.32 number
 : nvic-priority ( priority irq# -- )
  $E000E400 + c!
 ;

 : pa0.pulldown  ( -- )    \ PA0 is read temperature pushbutton, triggered when pulled high, so add pulldown.
   \ 3 2    1 0
   \ CNF0   MOD0
   \ 0 1    0 0    default input, floating
   \ 1 0    0 0    wanted   %10 constant cnf.input.pull.up.down
   2 bit GPIOA_CRL bic!    \ clear bit 2
   3 bit GPIOA_CRL bis!    \ set bit 3
   GPIOA_BRR_BR0           \ with F103, pulldown enabled setting output bit to low
 ;

 : pb13->output ( -- )
   \ 23 22  21 20
   \ CNF13  MODE13
   \ 0  1   1  1
   %11 GPIOB_CRH_MODE13    \ set both mode bits in output mode 50
   %11 GPIOB_CRH_CNF13     \ clear both bits = push/pull in output mode
```

```
;

: lmt01.power.on  ( -- )
  GPIOB_BSRR_BS13
;

: lmt01.power.off ( -- )
  GPIOB_BSRR_BR13
;

: lmt01-fault? ( -- ) count.lmt01 @ 0 = ;         \ if count.lmt01 >0, no error

: lmt01.interrupt.handler ( -- )
  line.12.mask                                    \ prevent line.12 retriggering until fully handled
  EXTI_PR_PR12                                    \ clear trigger on line 12
  1 count.lmt01 +!                                \ increment count.lmt01 by one
  line.12.unmask                                  \ line.12 can be retriggered now
;

: lmt01.interrupt.init ( #40 )
  1 AFIO_EXTICR4_EXTI12                           \ line PB12
  line.12.unmask
  EXTI_FTSR_TR12                                  \ falling trigger on line 12
  ['] lmt01.interrupt.handler irq-exti10 !        \ link exti10 ( to pb.interrupt.handler
  %100000000 NVIC_ISER1_SETENA                    \ enable enterrupt #40
;

: read.lmt01 ( lmt01 -- count )          \ read sensor, power on, delay, power off
  0 count.lmt01 !
  ms.counter.reset
  lmt01.power.on
  100 ms.delay
  lmt01.power.off
  lmt01-fault? if                        \ count window has ended, if no counts there has been an error.
```

```
     ." LMT01 FAULTY OR NOT PRESENT ! "
     else
     count.lmt01 @
     then
  line.0.unmask                          \ pb.interrupt may now be triggered again
;

: convert.count.to.degrees.c ( count -- degC )    \ convert count.lmt01 to deg C
  >s31.32                                         \ Temp (C) = ((count.lmt01/4096) *256) -50
  4096,0 f/ 256,0 f* 50,0 d-
;

: convert.count.to.degrees.f ( -- )
  convert.count.to.degrees.c
  1,8 f* 32,0 d+                         \ °F = °C * 1.8000 + 32.00

;

: C ( -- )                               \ Deg C print shortcut
  read.lmt01
  convert.count.to.degrees.c
  2 f.n ." C " cr
;

: F  ( c -- )                            \ Deg F print shortcut.
  read.lmt01
  convert.count.to.degrees.f
  2 f.n ." F " cr
;

: pb.interrupt.handler ( -- )
  line.0.mask                            \ prevent retriggering until fully handled
  EXTI_PR_PR0                            \ clear trigger
  C F cr                                 \ read temperatures
;

: pb.interrupt.init ( #6 )
```

```
   0 AFIO_EXTICR1_EXTI0                   \ line PA0
   line.0.unmask                          \ unmask line 0
   EXTI_RTSR_TR0                          \ rising trigger in line 0
   ['] pb.interrupt.handler irq-exti0 !   \ link exti0 to pb.interrupt.handler
   %1000000  NVIC_ISER0_SETENA            \ enable interrupt #6
 ;

 : interrupt.priorities ( -- )            \ highest number is lowest priority
   100 40   nvic-priority                 \ lmt01 sensor pulse detect
   200  6   nvic-priority                 \ pushbutton
 ;

 : init ( -- )
   72mhz
   interrupt.priorities
   systick.init
   lmt01.interrupt.init
   pb.interrupt.init
   pa0.pulldown
   pb13->output
   +usb
 ;

 init


\ Screenshots
\
\ c 35,68 C
\  ok.
\ f 96,23 F
\  ok.

\ press Push Button PA0
\
\ 35,68 C
\ 96,34 F
\
\ 35,68 C
\ 96,34 F
```

```
\
\ read.lmt01 . 1371  ok.  <-- raw count @ 35,68 C = 96,34 F
```

Play With Forth ?

This Image is a complete system, it contains the compiler and many internal Peripheral registers, see the ae9236b921613c07b1967f21ff20ee97.words4.txt file for a listing of what’s there.

Wonder what they all mean ? see the Dictionary.

You can read, write and examine registers, run programs, even commit them to Flash memory.

Note

to erase all the Flash except the default Mecrisp-Stellaris Forth system, enter “eraseflash” at the terminal. This will remove everything from this project including the USB, and only USART2 will work with a terminal. To reinstall everything just reflash the ae9236b921613c07b1967f21ff20ee97.bin imahe to your board.

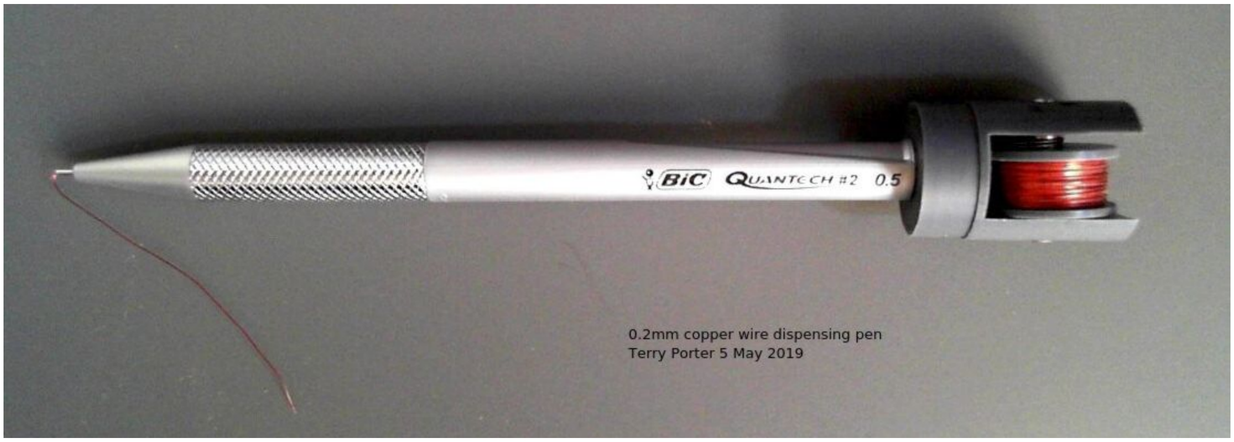
Prototyping

General descriptions of the methods I use for prototyping small Forth powered devices. Prototyping is usually limited to one or two units, and if a prototype was going to go into production, I would use a commercially made run of PCB’s instead.

Normally, a prototype and production unit have very little in common.

Hardware

Wiring



Wiring Pen

For analog and digital signal prototype wiring I handwire using 0.2mm polyurethane enameled copper wire (also called “magnet wire”) with the dispensing pen in the picture above. This was a off-the-shelf 0.5mm BiC lead pencil fitted with a former and holder I machined out of PVC. In the gap between former and holder is a small spring providing the friction necessary to prevent the former spinning and un-spooling the wire by accident.

The magnet wire is coated with polyurethane enamel which is easily removed by sliding a blob of solder on the tip of a soldering iron over the **end of the wire**. Unlike larger diameter 0.5mm Teflon coated wire wrap wire the polyurethane enamel removal is clean and tidy without having to scrape of melted plastic afterwards. The polyurethane enameled copper wire also holds its shape where the alternative is slightly springy and much bulkier.

Note

Insulation removal other than at the end of the wire is difficult and I don’t generally do that, rather I use point to point wiring rather than a continuous string when connecting a number of pins together.

Wire Source

I use the wire from a old transformer winding, old electromagnet loudspeaker wire is also a great source. If buying online, check the temperature rating of the insulation which should be 300C or less if possible. If it isn’t then the wire won’t be suitable, and if the vendor doesn’t know, search elsewhere.

Iron Temperature

350C of soldering temperature is recommended for this process.

Visual Difference Between Suitable and Unsuitable Wire ?

None, they all look alike.

REASONS FOR FAILURE

- Iron tip not hot enough
- Wire insulation is wrong type. These days it’s difficult to ensure the parts you buy are what the vendor claims them to be. If this method didn’t work, I wouldn’t have been using it since 1976 and writing about it here.
- Trying to remove insulation in middle of wire and not at the end

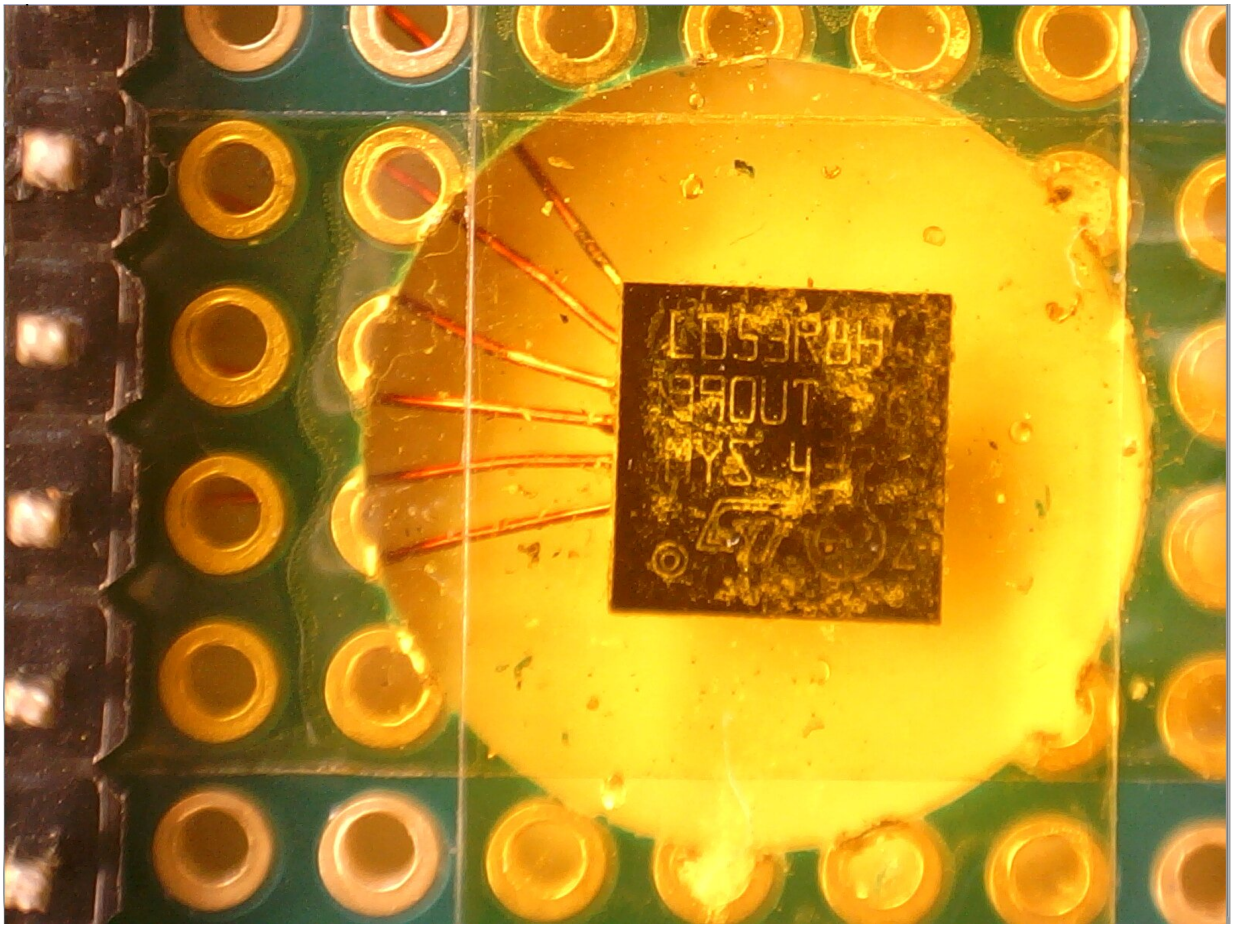
Examples

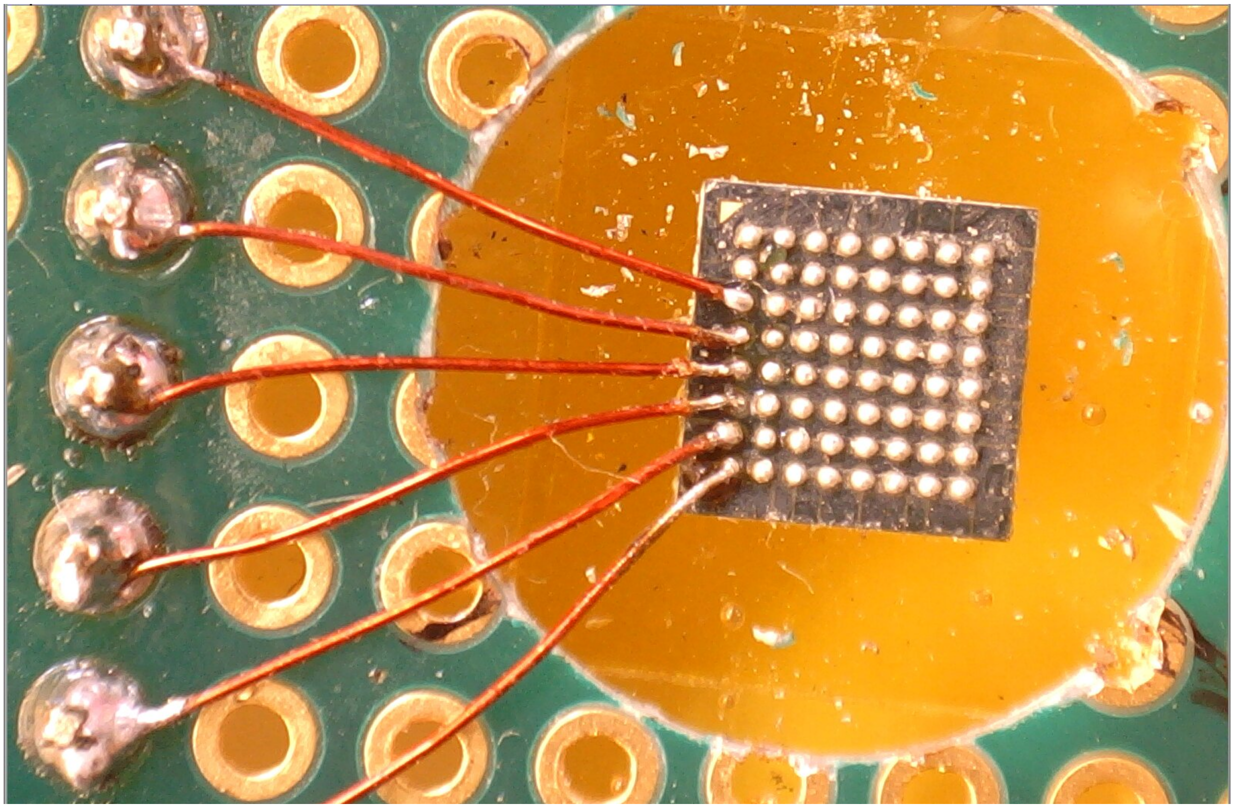
My Work

Wire was spooled via my ‘wiring pen’ above and hand soldered. This project is unfinished at the time of the picture.

STM32L053

- material: 0.1” pitch matrix board from a STM32 Discovery Board Kit
- Wire: 0.2mm polyurethane enameled copper wire which is the smallest I have.
- Hole drilled board for SMT chip
- Kapton tape used to retain the chip in hole


CDS3R8H
39OUT
MYS 4



Copper Rivets

I’m experimenting with home made, hand operated, tool formed, copper rivets to use with BGA and QFN MCU’s, hand spooled wire, blank Teflon pcb, Kapton tape and hand soldering.

These are some pictures of my third prototype rivet. It is approximately 4mm by 2.3mm in size and made from solid copper wire recovered from a old transformer.

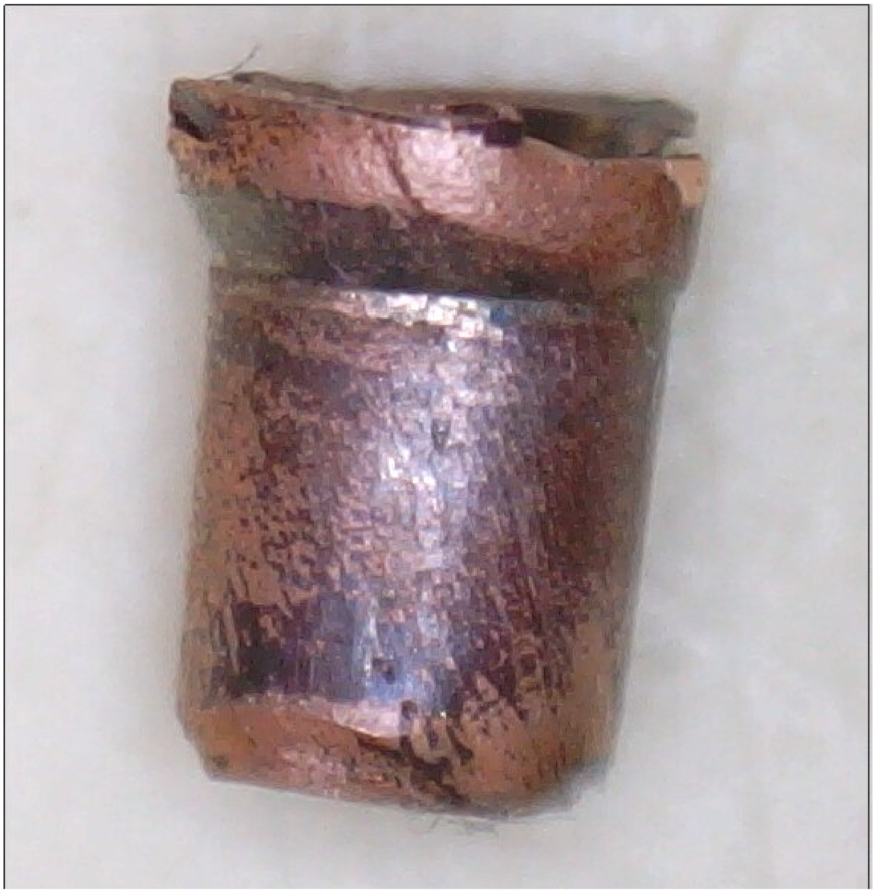
The rivets will be used as smt, leaded components and multicore wire anchor/solder points, with some rivets being drilled thru like a via on a etched pcb.

Note

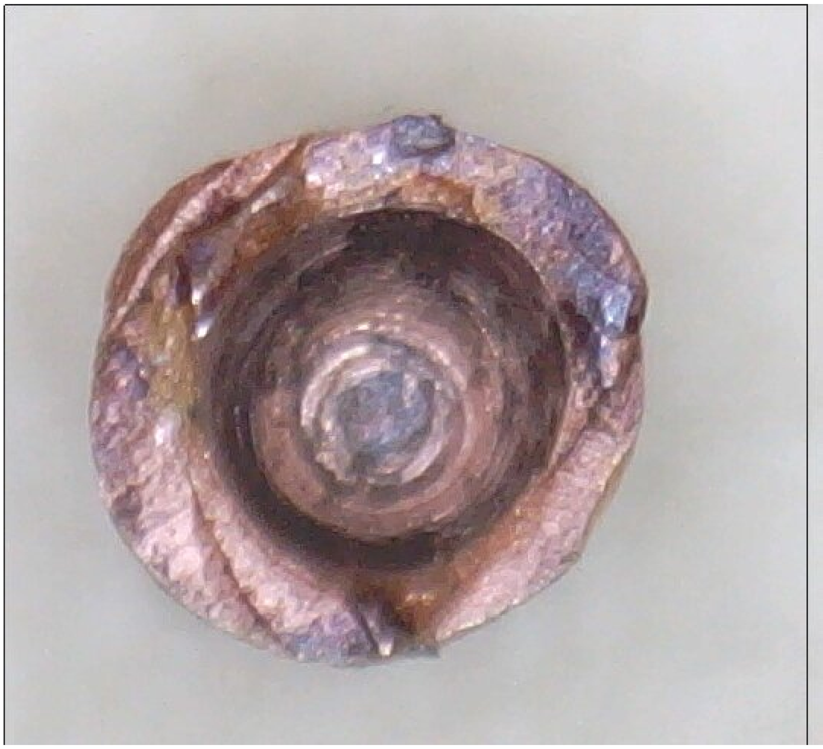
All pics are from a USB microscope.

Rivet side view: the shaft is 2.3 mm in diameter

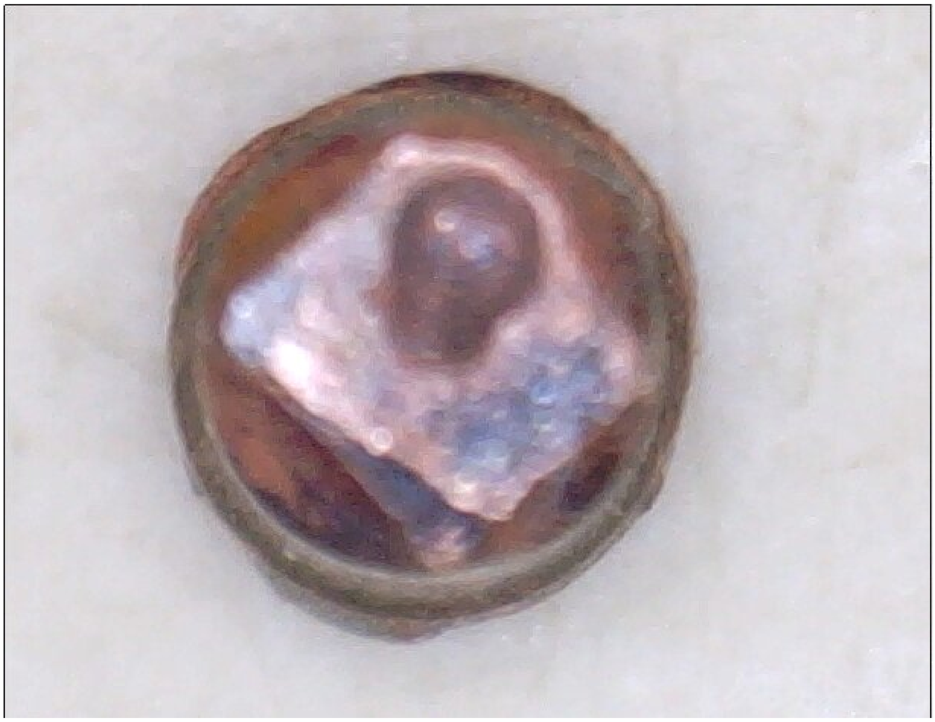
This image is 30.5 times larger than life size.



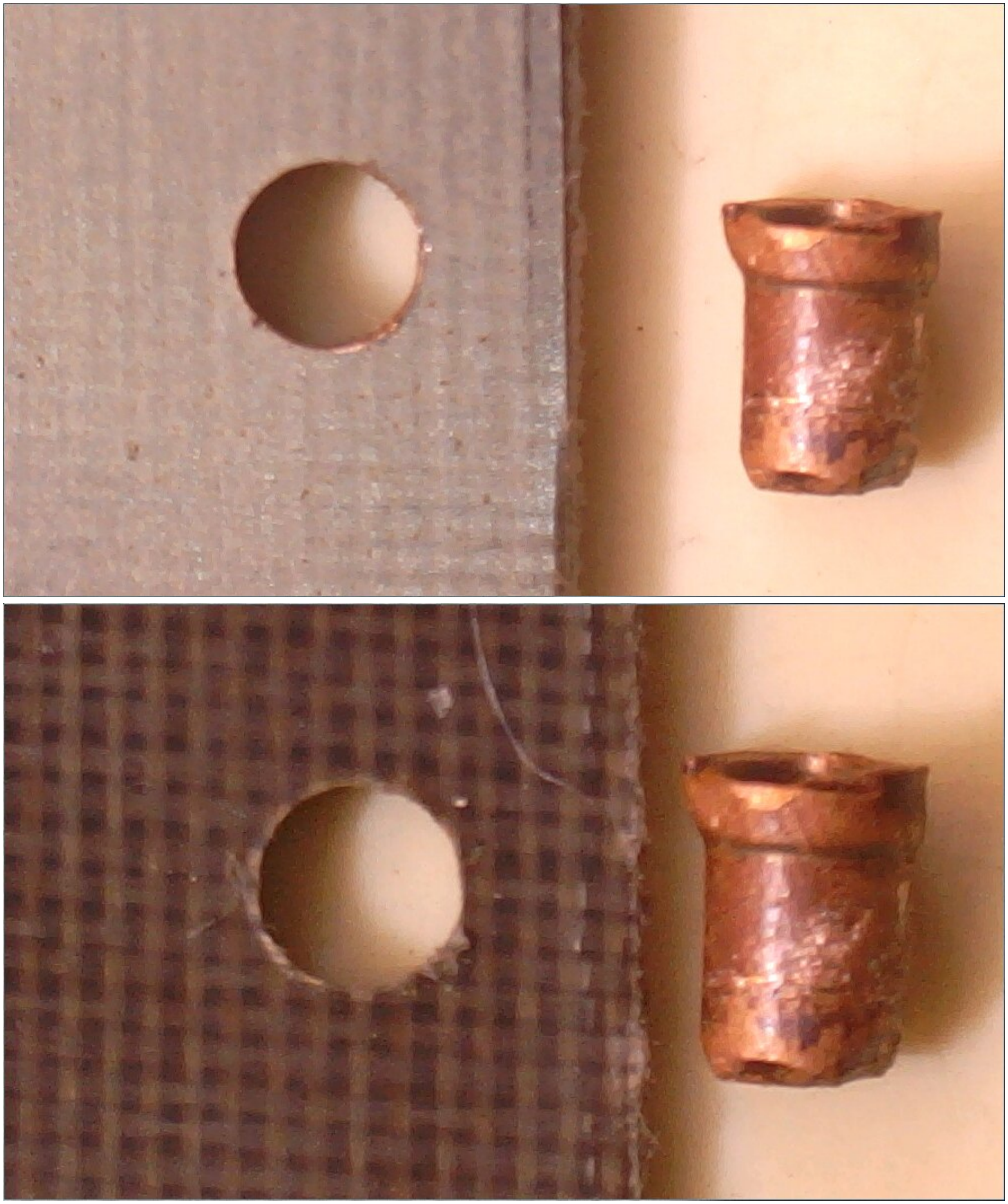
Rivet top view



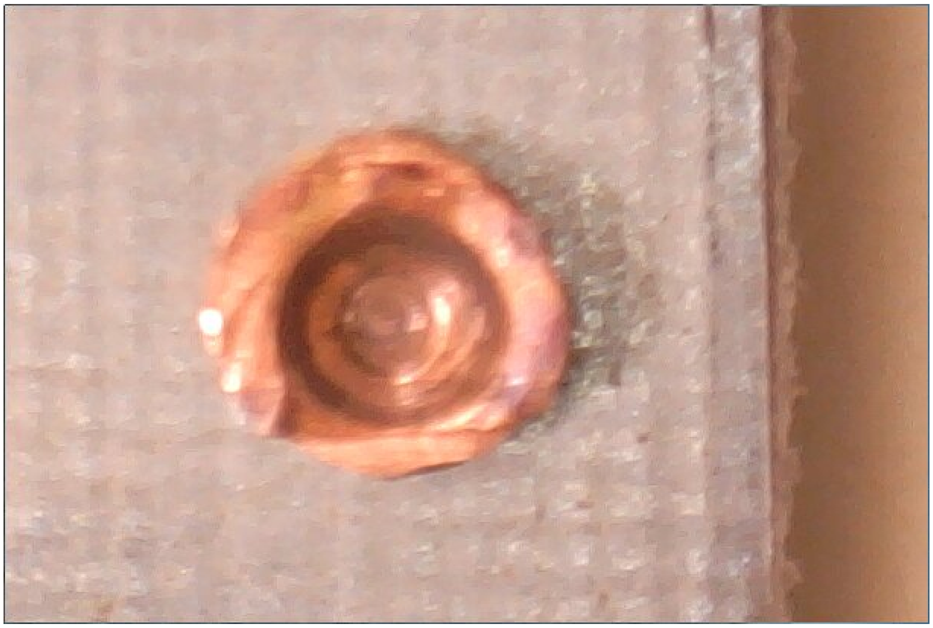
Rivet focus bottom of top lip

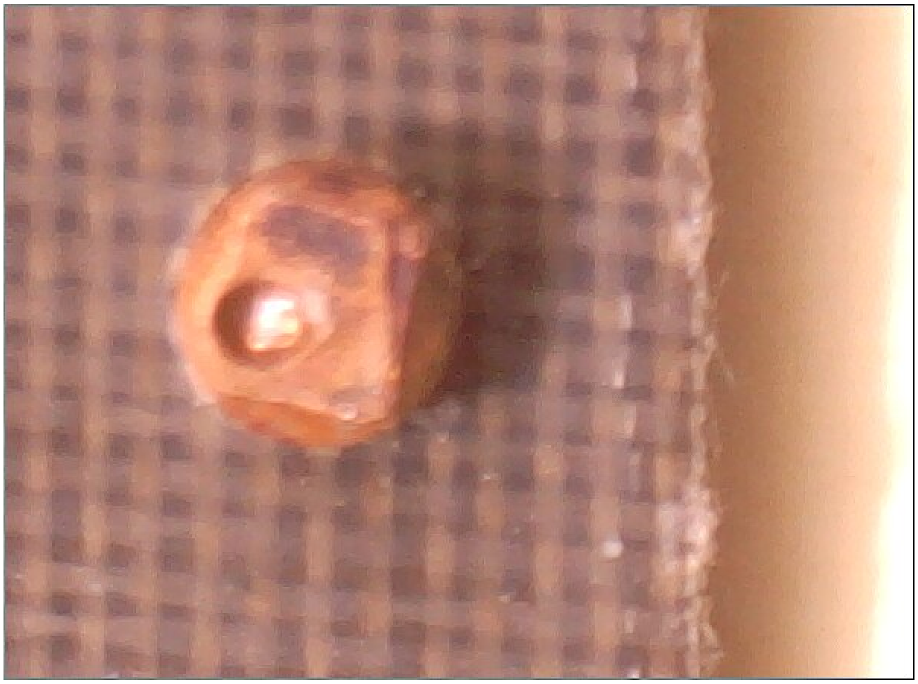


Rivets next to 2.3mm hole in PCB



Rivets in 2.3mm hole in PCB





Rivet Flattened At Each End, Drilled (1.2mm) And Soldered

Liquid flux used, same as I use with SMT soldering. Then a quick rub with a fine wire brush to remove the gunk, hence the fine scratches. I’m testing the basic functionality at this point and not looking to win any beauty contests.

This is 1mm thick teflon composite pcb material from a 5.6GHz antenna and the rivet is a bit too close to the edge which you can see has a small bulge from the rivet expanding as it was pressed. Next time I’ll anneal the rivet, fit it further from a edge and not squish it so much.

Rivet Edge View

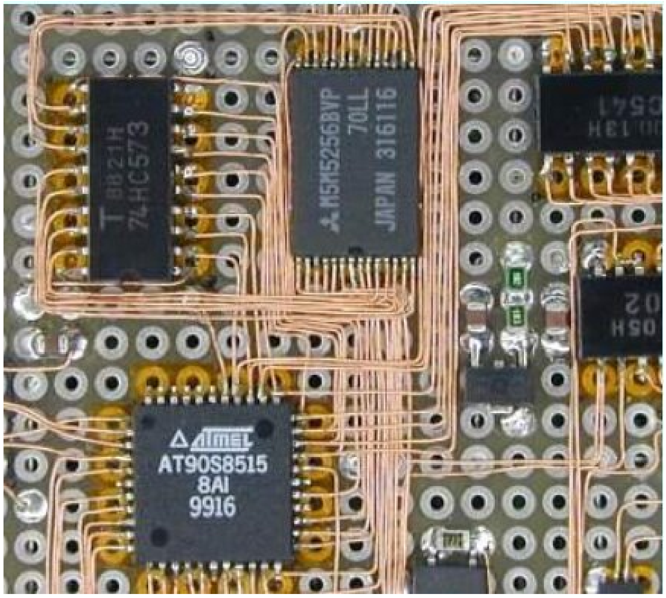
Elm Chan

A beautiful example of precision hand wiring and soldering

This example taken from http://elm-chan.org/docs/wire/wiring_e.html

Note

If this method didn’t work, then the picture below (by Elm Chan) wouldn’t exist.



How to prepare thicker wire with much higher temperature enamel for soldering ?

- Using a gas lighter (cigarette) or a match hold the small flame over the end of the wire which will blacken as the enamel burns off
- Lightly sand the black off with fine sandpaper or use a knife, my favorite is my “three cornered scraper”
- Solder coat the end with your iron as usual

Source of suitable wire

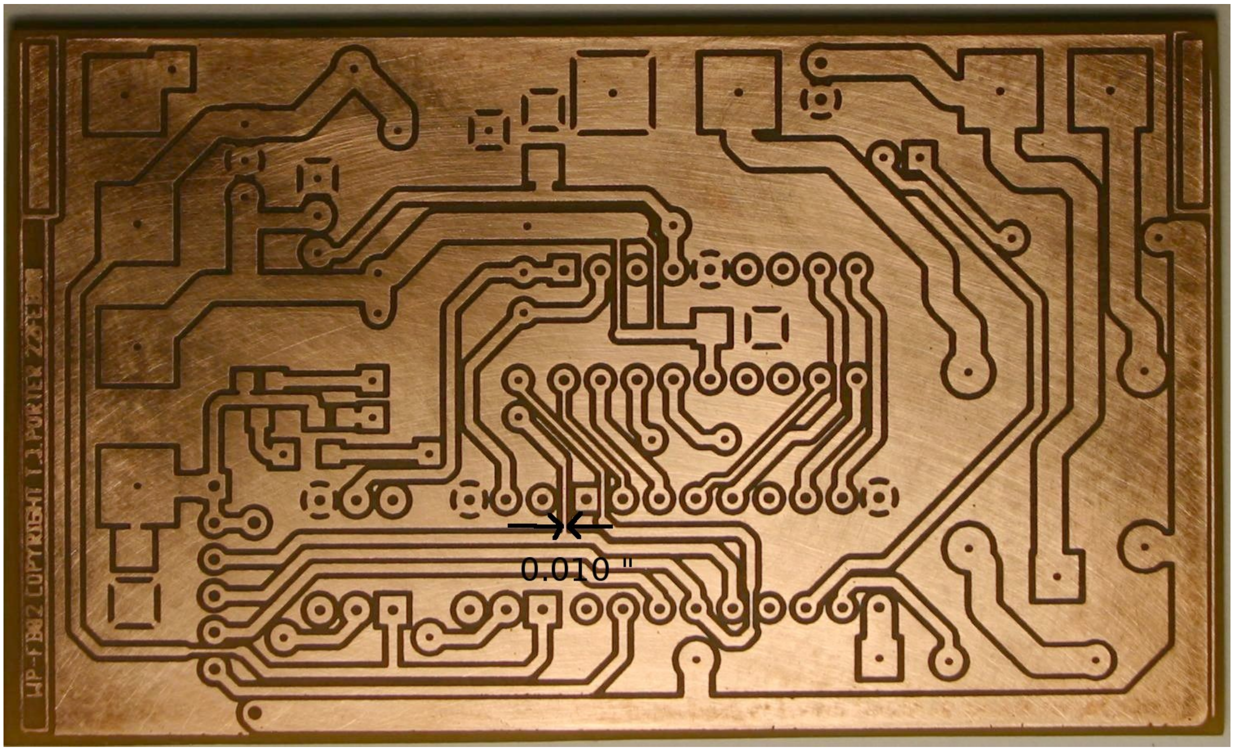
Be sure to get the 155 C rated type

URL	INSULATION MELT TEMP	SIZE (mm)	COMMENTS
https://www.scientificwire.com/acatalog/ec-wire.html	155 - 180 C	0.180 - 1.25	green, red, and orange, 6kg minimum £66 + shipping

PCB

After a prototype has been completed and tested using the wiring method I then make up a PCB which I do in-house and looks something like the picture below.

This hand etched copper pcb sat in my leather wallet for about 10 years after I made it. I gave it a quick sanding as it was a bit green and corroded then took this picture.



Methods

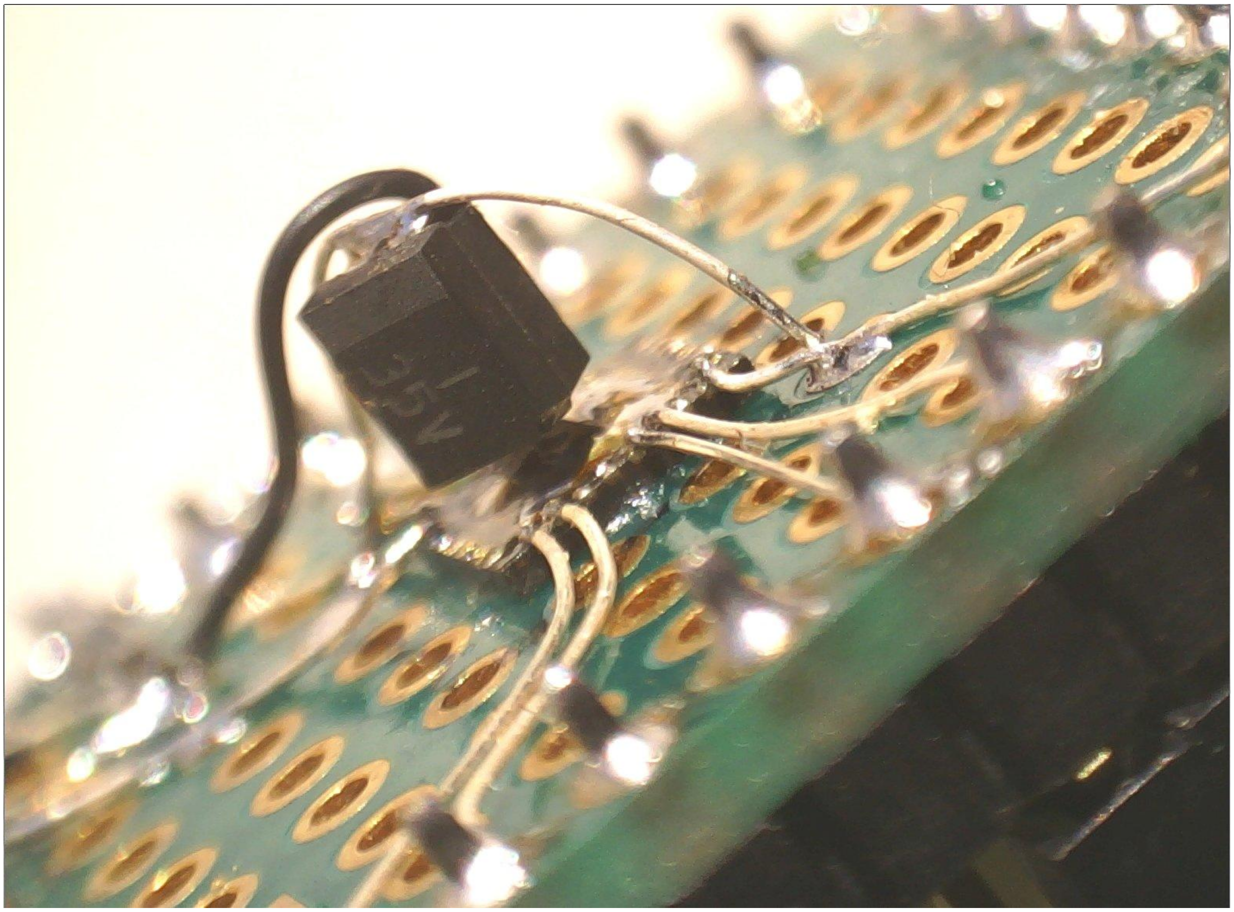
I use simple techniques applied by hand for my single sided prototype pcbs.

Action	**Description**
Schematic Capture Software	gEDA Gschem (GPL) http://www.geda-project.org
PCB Cad Layout Software	gEDA PCB (GPL)
Copper	With photo resist applied
Artwork	Laser printed on Drafting Film paper

Action	Description
Exposure	Light Box with Black Light fluro tubes
Etchant	Ferric Chloride heated and agitated
Drilling	Dremel high speed drill on a hand operated stand
Drill Bits	Carbide

A deadbugged STM32F051, hand soldered

Running Mecrisp-Stellaris !



Radiation Hardened CPU

Source: https://wikivisually.com/wiki/RTX2010

> Due to the extensive development and testing required to produce a radiation-tolerant design of a microelectronic chip, radiation-hardened chips tend to lag behind the most recent developments. Environments with high levels of ionizing radiation create special design challenges, a single charged particle can knock thousands of electrons loose, causing electronic noise and signal spikes. In the case of digital circuits, this can cause results which are inaccurate or unintelligible and this is a particularly serious problem in the design of satellites, military aircraft, nuclear power stations, and nuclear weapons. In order to ensure the operation of such systems, manufacturers of integrated circuits. The resulting systems are said to be rad-hardened, rad-hard, or hardened, cosmic rays come from all directions and consist of approximately 85% protons, 14% alpha particles, and 1% heavy ions, together with x-ray and gamma-ray radiation. Most effects are caused by particles with energies between 0.1 and 20 GeV, the atmosphere filters most of these, so they are primarily a concern for spacecraft and high-altitude aircraft.

Solar particle events come from the direction of the sun and consist of a flux of high-energy protons and heavy ions. Van Allen radiation belts contain electrons and protons trapped in the geomagnetic field, the particle flux in the regions farther from the Earth can vary wildly depending on the actual conditions of the Sun and the magnetosphere. Due to their position they pose a concern for satellites, secondary particles result from interaction of other kinds of radiation with structures around the electronic devices. Nuclear reactors produce gamma radiation and neutron radiation which can affect sensor, in case of a nuclear war they pose a potential concern for all civilian and military electronics. Chip packaging materials were a source of radiation that was found to be causing soft errors in new DRAM chips in the 1970s. Traces of radioactive elements in the packaging of the chips were producing alpha particles and these effects have been reduced today by using purer packaging materials, and employing error-correcting codes to detect and often correct DRAM errors.

Two fundamental damage mechanisms take place, Lattice displacement, caused by neutrons, alpha particles, heavy ions, components certified as ELDRS free, do not show damages with fluxes below 0.01 rad/s =36 rad/h. Ionization effects are caused by charged particles, including the ones with energy too low to cause lattice effects, the ionization effects are usually transient, creating glitches and soft errors, but can lead to destruction of the device if they trigger other damage mechanisms. Photocurrent caused by ultraviolet and x-ray radiation may belong to category as well. Gradual accumulation of holes in the layer in MOSFET transistors leads to worsening of their performance. The end-user effects can be characterized in several groups, Neutron effects and this leads to an increase in the count of recombination centers and deep-level defects, reducing the lifetime of minority carriers, thus affecting bipolar devices more than CMOS ones. Bipolar devices on silicon tend to show changes in electrical parameters at levels of 1010 to 1011 neutrons/cm², the sensitivity of the devices may increase together with increasing level of integration and decreasing size of individual structures

The RTX2010

Manufactured by Intersil is a radiation hardened stack machine microprocessor which has been used in numerous spacecraft. Characteristics

It is a two-stack machine, each stack 256 words deep, that supports direct execution of Forth. Subroutine calls and returns only take one processor cycle and it also has a very low and consistent interrupt latency of only four processor cycles, which lends it well to realtime applications.

RTX2010 History

In 1983, Chuck Moore implemented a processor for his programming language Forth as a gate array. As Forth can be considered a dual stack virtual machine, he made the processor, Novix N4000, as a dual-stack machine. In 1988, an improved processor was sold to Harris Semiconductor, who marketed it for space applications as the RTX2000.

Example spacecraft that use the RTX2010

- Advanced Composition Explorer (ACE)
- NEAR/Shoemaker
- TIMED
- Rosetta’s lander - Philae

Philae Lander

Philae is controlled by 2 RTX2010 CPUs and a further 8 control the Experiments. 13 CPUs (10* RTX2010, 1* ADSP-21020, 2* 80C3x) in total.

Random Access Memory

Free Ram ?

Ram starts at $2000000 and grows upwards to the physical limit. My STM32F051 has 8KB Ram therefore the maximum is at $20002000

Mecrisp will use a certain amount ($38C or 908 decimal bytes in this example).

Find The Start of Free Ram

```
compiletoram here hex. 2000038C  ok.
```

Create The Word “freeram”

```
: freeram ( -- )
  flashvar-here 4 -
  compiletoram here
  - . ." Bytes "
;
```

“freeram” Word Dictionary Entry

```
Address: 2000038C Link: 000000C0 Flags: 00000000 Code:
2000039A Name: freeram
```

Note

Tick(‘) also shows markers memory location

```
' freeram hex. 2000039A  ok.
```

Free Ram Start (after “freeram” has been created)

```
compiletoram here hex. 200003D0  ok.
```

Find The End of Free Ram

```
flashvar-here 4 -
```

A Word to Find Free Ram

“freeram” already created above

```
: freeram ( -- )
  flashvar-here 4 -
  compiletoram here
  - . ." Bytes "
;
```

```
freeram

freeram 7028 Bytes  ok.
```

RA Kernel

The RA (Register Allocator) Kernel contains an analytical compiler which keeps track of the top five stack elements and maps them to registers whenever possible.

It is Matthias latest Mecrisp-Stellaris development line and available for about 35 of the MCUs in the supported hardware list, but can be enabled at compile time for all MCUS when building the Core binary.

It features:

- constant folding
- inlining
- tail-call optimisations

RA Register Map

```
r0: Free scratch register            ( Saved on interrupt entry by hardware )
r1: Free scratch register            ( Saved on interrupt entry by hardware )
r2: Free scratch register            ( Saved on interrupt entry by hardware )
r3: Free scratch register            ( Saved on interrupt entry by hardware )

r4: Inner loop count                 ( Needs Push and Pop when used otherwise )
r5: Inner loop limit                 ( Needs Push and Pop when used otherwise )

r6=TOS: Top-Of-Stack                 ( Stack design is interrupt safe )
r7=PSP: Parameter Stack Pointer      ( Stack design is interrupt safe )
```

```
r8:  Unused
r9:  Unused
r10: Unused
r11: Unused
r12: Unused                          ( Saved on interrupt
entry by hardware )

r13=SP: Return Stack Pointer
r14=LR: Link Register
r15=PC: Program Counter, always odd
```

RA Bitexp Example

```
: bitexp ( u -- u )
 \ Returns an integer value equivalent to
 \ the exponential. For numbers > 16,
 \ bitexp(x) approx = 2^(x/8 + 1)
 \ B(E(x)) = x for 16 <= x <= 247.

 dup 247 u>  \ Overflow ?
 if drop $F0000000
 else
   dup 16 u<= if 1 rshift
              else
                dup ( u u )
                7 and 8 or ( u b )
                swap ( b u )
                3 rshift 2 - lshift
              then
 then
 1-foldable ;
```

See bitexp

Note that it compiles bitexp without any stack movements at all

```
00006BA2: 2EF7  cmp r6 #F7
00006BA4: B500  push { lr }
00006BA6: D902  bls 00006BAE
00006BA8: 26F0  movs r6 #F0
```

```
00006BAA: 0636  lsls r6 r6 #18
00006BAC: E00C  b 00006BC8
00006BAE: 2E10  cmp r6 #10
00006BB0: D801  bhi 00006BB6
00006BB2: 0876  lsrs r6 r6 #1
00006BB4: E008  b 00006BC8
00006BB6: 0033  lsls r3 r6 #0
00006BB8: 2007  movs r0 #7
00006BBA: 4003  ands r3 r0
00006BBC: 2008  movs r0 #8
00006BBE: 4303  orrs r3 r0
00006BC0: 08F6  lsrs r6 r6 #3
00006BC2: 3E02  subs r6 #2
00006BC4: 40B3  lsls r3 r6
00006BC6: 461E  mov r6 r3
00006BC8: BD00  pop { pc }
```

Convert your Classic Kernel to RA

Dive into the core assembly code.

Activate RA compiler optimisations by increasing the reserved core size to 0x5000 (up from the 0x4000 of the classic cores) and by adding an assembler switch:

```
.equ registerallocator, 1
```

Rebuild your Kernel.

Classic Mecrisp-Stellaris

- Does not have the RA compiler or register mapping
- lacks automatic inlining
- Won’t run (excepting mecrisp-stellaris-2.3.6-hook-find) Manfred Mahlow’s VOCS, a collection of source code modules to add wordlists, vocabularies, vocabulary prefixes and classes (for context switching objects) to Mecrisp-Stellaris Forth.
- Lacks the roll and -roll Words

Mecrisp-Stellaris Deeper Configuration

There are a few deeper configuration values in mecrisp-stellaris-2.4.X/mecrisp-stellaris-source/common/forth-core.s. Have a look there !

You can increase the stack sizes in that file. For default, they are set to 64 elements each

Rants

Here is where I try and explain why some things BUG ME.

Note

Disclaimer: I may not be right, I may not be wrong, these are my opinions and no one elses.

I’m inspired by the epic rants of ttm richter

I met TTM (Michael) Richter on IRC a few years back. A Canadian of Teutonic roots who lives in China, he’s a wordsmith, teacher and programmer who seems to love all types of software (including Forth), has strong opinions and can express them in the most entertaining ways. I have Michael to thank for suggesting the Fossil Software Code Management system and many other helpful tips that have made my own software journey easier and smarter.

Rant about github “linguist” program failures

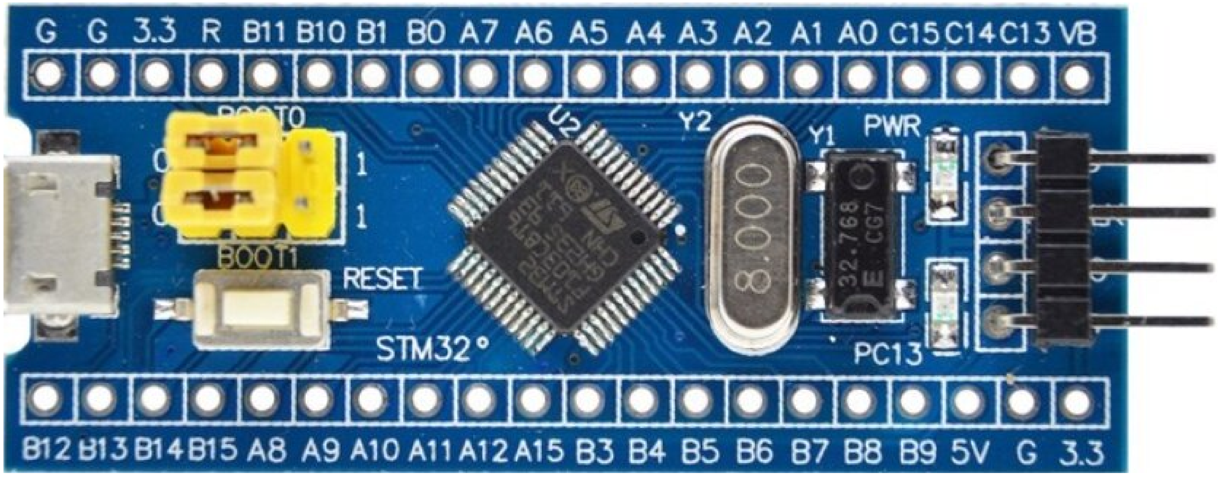
```
Jesus!
Could you please just SHUT THE FUCK UP ABOUT MAKE?!

The issue here is how Linguist sucks so strongly that it can
extract matter from galactic black holes. This is an issue
which has been outstanding
for over a year ON THIS SINGLE FUCKING LANGUAGE ALONE. I'm
sure if we scanned other PRs we'd find more that are even
older. (Like .pl and Prolog which
I highlighted above.) The code, as it is now, IS NOT CORRECT.
The code, as it has always been, HAS NEVER BEEN CORRECT. The
```

```
code, IS DEFECTIVE BY ITS VERY DESIGN.

If you really do believe that correctness is paramount ALL of
your diarrhetic verbiage about performance is disingenuous,
meaningless drivel. (It's meaningless
drivel if you don't believe that correctness is paramount,
but at least it's honest diarrhetic verbiage.) So which is
it? You can't have it both ways.
```

[ttmrichter]

QUICK, GRAB A BARGAIN

Blue Pill PCB


G G 3.3 R B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VB
BOOT0
BOOT1
RESET
STM32
Y2
Y1
PWR
8.000
32.768
PC13
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V G 3.3

A BARGAIN?

I just don’t get people who think that the “Blue Pill”, a “development board” based around the STM32F103 from China for $1.50 is a bargain.

I can’t even buy a cup of coffee for under $5 nowadays.

When I was in my early 20’s, a National Pace 16 bit CPU was $270. A new 8 bit Motorola 6800 was about $170, and both of those CPU’s needed extensive support devices because they had zero peripherals. When the 6502 (with a limited 8 bit index register) came out for $15 it was lapped up by generations of people who also thought it was a bargain.

The STM32F103 is a *OLD chip* based on the first generation of Cortex M3 chips announced in 2004, and manufactured since at least 2005. They have a lot of known problems.

A much better choice for new projects are either STM32F0, F0+ series or STM32F4 series. The prices are about the same for comparable parts.

Breaking Copy Protection

Your firmware isn’t safe either in this chip, see https://blog.zapb.de/stm32f1-exceptional-failure/

Then there is also the **COUNTERFEIT** issue …

Counterfeit

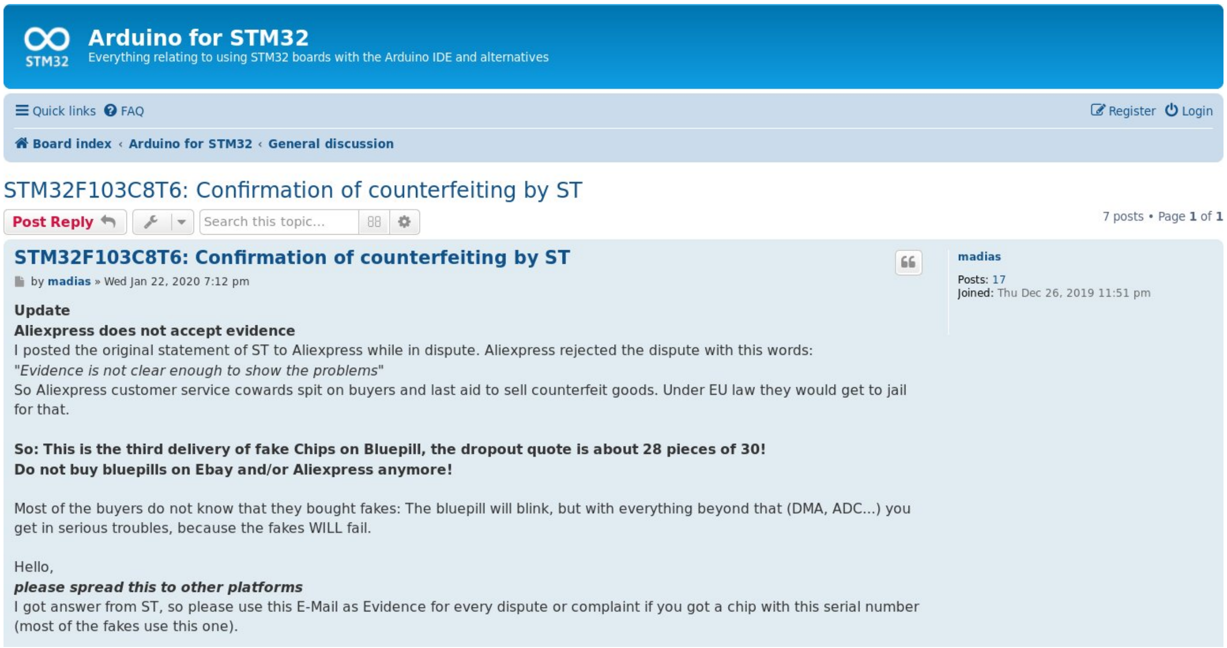
https://www.stm32duino.com/viewtopic.php?f=7&t=127

A Chinese engineers attitude about what I call ‘counterfeit’.

> Part numbers are not generally considered trademarks here. If I produce a transistor that adheres to all 2n3904 specifications, I can sell it as a 2n3904, even if some characteristics not on the datasheet are a bit different. Similarly if I make a chip that functions just like a STM32 and runs most code unmodified (except maybe for very few corner cases), it’s generally considered a-ok to sell it as a STM32. The generics mindset here is strong, and it’s not considered ok for one manufacturer to monopolize a part number, especially when compatible substitutes are produced. “STM32” is a generic name for a MCU with

these peripherals in this layout, just like U.FL is a generic name for a connector with these dimensions. If that’s a problem for you just buy from the big distributors.

Aliexpress refuse evidence and refund


Arduino for STM32
Everything relating to using STM32 boards with the Arduino IDE and alternatives
Quick links
FAQ
Register
Login
Board index ‹ Arduino for STM32 ‹ General discussion
Post Reply
Search this topic…
7 posts • Page 1 of 1

STM32F103C8T6: Confirmation of counterfeiting by ST

STM32F103C8T6: Confirmation of counterfeiting by ST

by **madias** » Wed Jan 22, 2020 7:12 pm

madias
Posts: 17
Joined: Thu Dec 26, 2019 11:51 pm

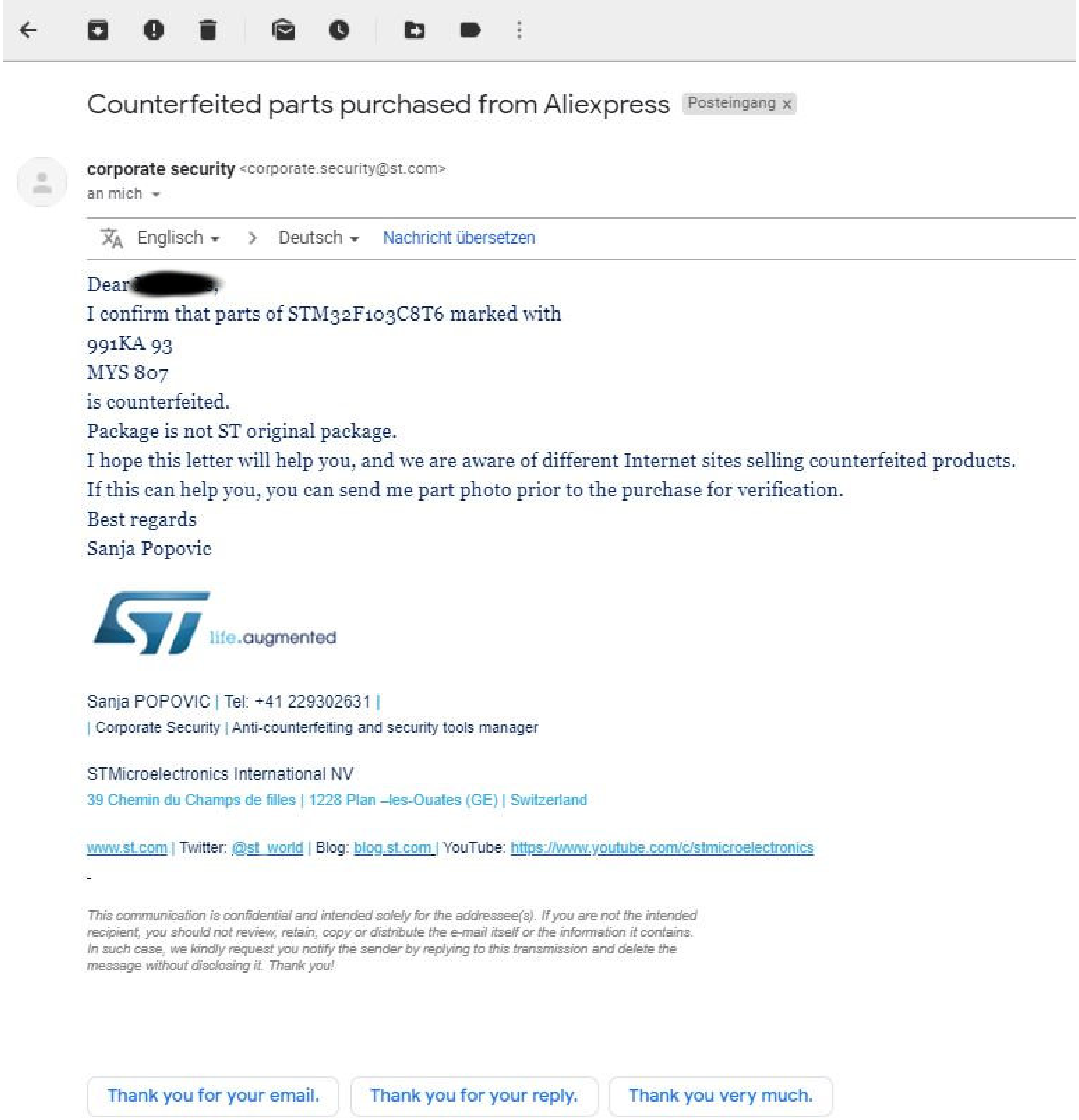
Update
Aliexpress does not accept evidence
I posted the original statement of ST to Aliexpress while in dispute. Aliexpress rejected the dispute with this words:
"*Evidence is not clear enough to show the problems*"
So Aliexpress customer service cowards spit on buyers and last aid to sell counterfeit goods. Under EU law they would get to jail for that.

So: This is the third delivery of fake Chips on Bluepill, the dropout quote is about 28 pieces of 30!
Do not buy bluepills on Ebay and/or Aliexpress anymore!

Most of the buyers do not know that they bought fakes: The bluepill will blink, but with everything beyond that (DMA, ADC...) you get in serious troubles, because the fakes WILL fail.

Hello,
please spread this to other platforms
I got answer from ST, so please use this E-Mail as Evidence for every dispute or complaint if you got a chip with this serial number (most of the fakes use this one).

ST Micro letter verifies counterfeit



Counterfeited parts purchased from Aliexpress

corporate security <corporate.security@st.com>
an mich ▾

Dear [redacted],
I confirm that parts of STM32F103C8T6 marked with
991KA 93
MYS 807
is counterfeited.
Package is not ST original package.
I hope this letter will help you, and we are aware of different Internet sites selling counterfeited products.
If this can help you, you can send me part photo prior to the purchase for verification.
Best regards
Sanja Popovic

life.augmented

Sanja POPOVIC | Tel: +41 229302631 |
| Corporate Security | Anti-counterfeiting and security tools manager

STMicroelectronics International NV
39 Chemin du Champs de filles | 1228 Plan –les-Ouates (GE) | Switzerland

www.st.com | Twitter: @st_world | Blog: blog.st.com | YouTube: https://www.youtube.com/c/stmicroelectronics

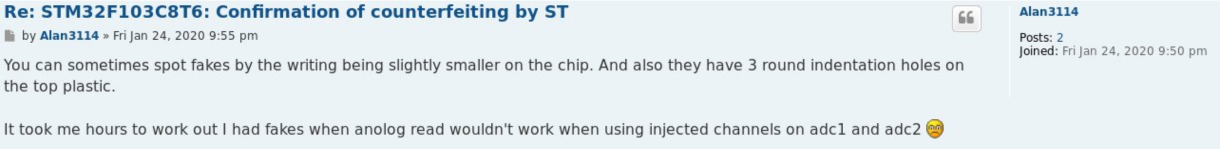
-

This communication is confidential and intended solely for the addressee(s). If you are not the intended recipient, you should not review, retain, copy or distribute the e-mail itself or the information it contains. In such case, we kindly request you notify the sender by replying to this transmission and delete the message without disclosing it. Thank you!

Counterfeit Incompatabilities

Anolog Read Issue

https://www.stm32duino.com/viewtopic.php?p=865&sid=4691deba3946368120f63923a877f10a#p865



Re: STM32F103C8T6: Confirmation of counterfeiting by ST

by **Alan3114** » Fri Jan 24, 2020 9:55 pm

You can sometimes spot fakes by the writing being slightly smaller on the chip. And also they have 3 round indentation holes on the top plastic.

It took me hours to work out I had fakes when anolog read wouldn't work when using injected channels on adc1 and adc2 🙁

Alan3114

Posts: 2
Joined: Fri Jan 24, 2020 9:50 pm

See also

[What chip is actually in your Blue Pill and how much Flash does it have ?]()

See also

[STM32F103C8 V/S GD32VF103Cx]()

See also

[Blue Pill Starter Kit]()

Some STM32F103 Problems

```
Peoblem                          Limitation
========                         ==========
CAN Bus                          Shares memory with the usb
hardware, which means that usb and canbus cannot be enabled
at THE SAME TIME
Serial Comms                     NO AUTOBAUD
USB                              Shares memory with the
canbus hardware, which means that usb and can cannot be
enabled at THE SAME TIME.
USB                              must use a CRYSTAL unlike
later stm32 chips which use only a internal RC clock for USB
Digital to Analogue Converter    DOESN’T have one
GPIO I/O port registers          Have to be accessed as 32-
bit words, half-word or byte accesses are NOT allowed. Very
CONFUSING to configure.
GPIO I/O port register           Very CONFUSING to configure
because of GPIOx_CRH and GPIOx_CRL. Cortex-M0 is MUCH more
logical making config trivial.
```

```
I2C                              There are SIX pages of
ERRATA for the I2C peripheral
DFU                              NO DFU bootloader (for
flashing via USB)
DEBUG                            Debug registers CANNOT be
read by user software. Workaround: none. Want to read the
Device ID via a program ? too bad!
```

Known Blue Pill Issues

- R10 is often the wrong value. The USB standard require a 1.5k pull-up resistor on D+ , but this board is known to have a wrong value with either a 10K resistor or a 4.7k resistor fitted. Google “R10 Blue Pill” for more information.
- Header pins are rubbish and they rust easily. Experienced Blue Pill users throw them away and use gold plates header pins, which doubles the cost of the $2 board!
- USB socket breaks off easily being poorly attached. Experienced Blue Pill users add extra solder etc.
- Fake MCU. These can be:
    - Relabelled STM units with less memory, i.e. 32kB or even 16kB Flash
    - Clones such as the GD32F103 or CKS32F103C8T6 (whatever that is)
    - Possibly recycled e-waste chips as billions of the STM32F103’s have been made.

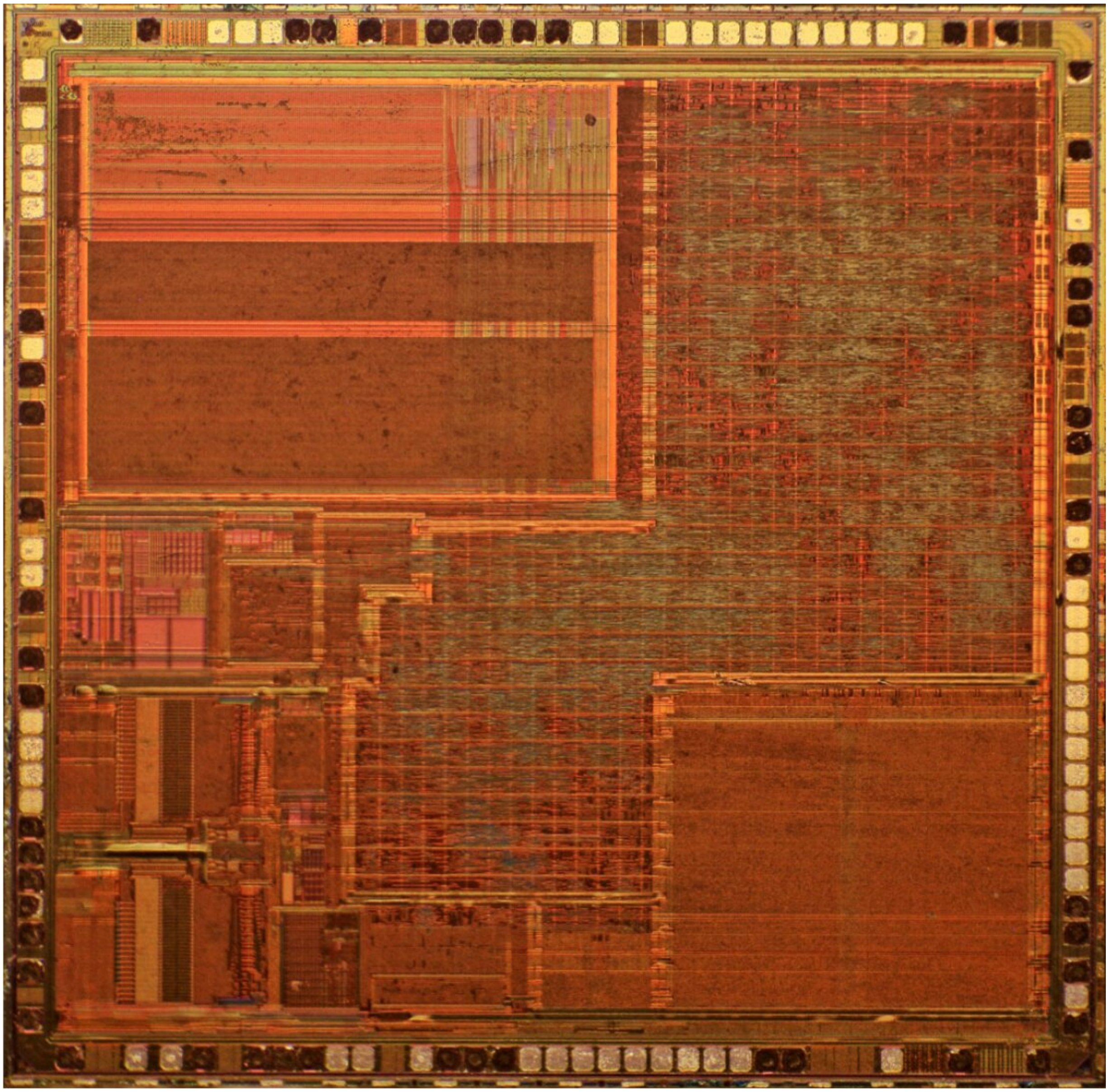
Now just think for a minute, a new STM32F103RBT6 is today is priced at $4.525 each in quantities of 100 from Arrow.com, so how is it possible that a Chinese vendor can sell you one in a PCB with switch, option jumpers, USB socket, two crystals and a voltage regulator for $1.50, what’s wrong with this picture ?

This is one way …

This is another …

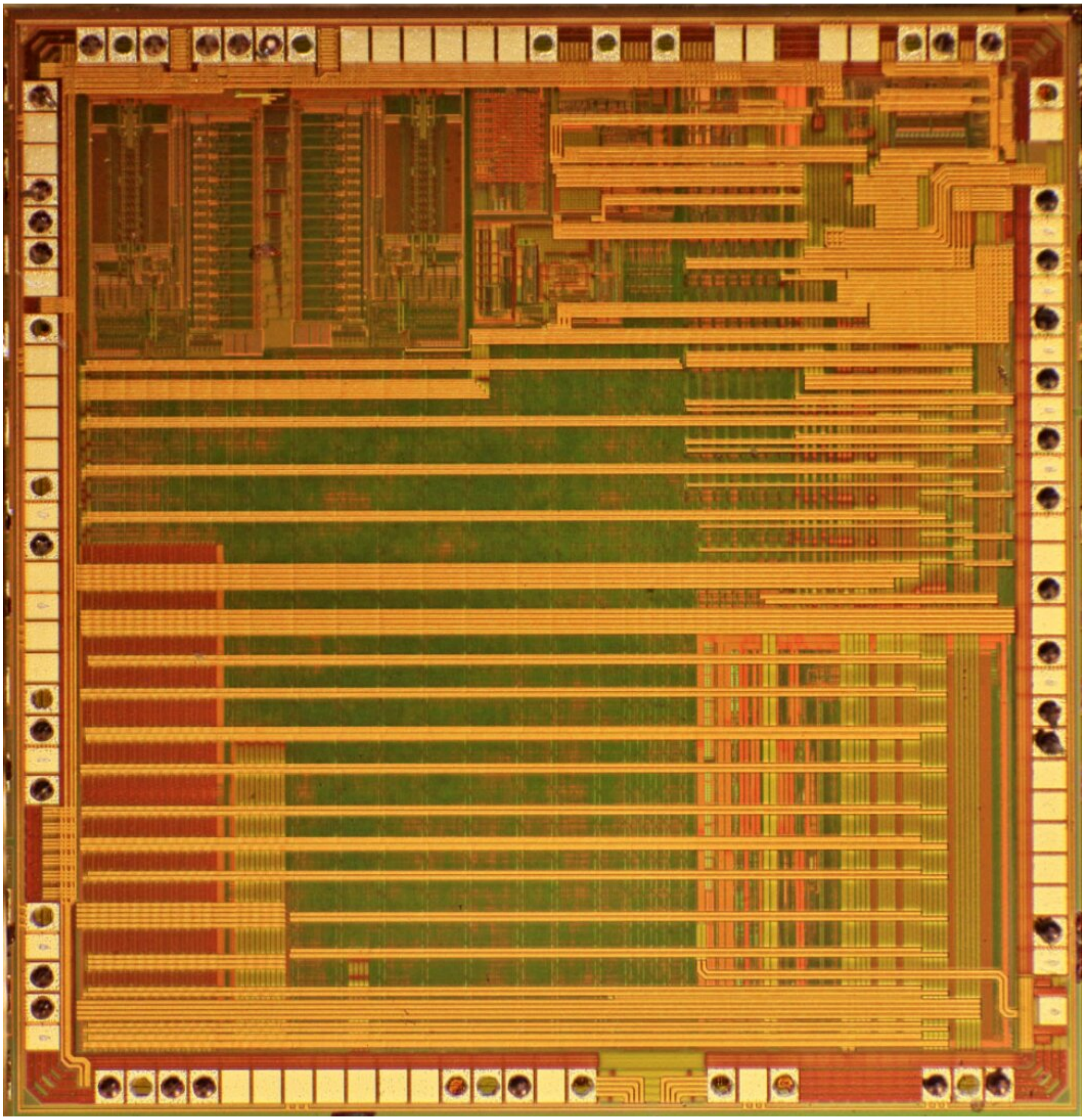
Genuine STM32F103C8 Decapped

Copied from https://www.richis-lab.de/STM32.htm



CKS32F103C8T6 Decapped

The “CKS32F103C8T6” is a fake it is not even listed on the CKS product website. Its a fake because it counterfeits the STM32 ID’s and name internally. Copied from https://www.richis-lab.de/STM32.htm



Using the CKS32F103C8T6 isn’t always straightforward either as this user found:

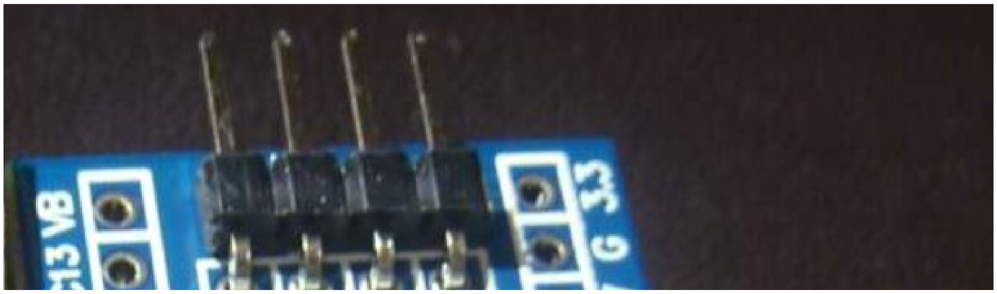
```
I`ve got chinese clone CKS32f103c8t6 Blue Pill board with ST-LINK V2 (cheap clone again).I`m able to flash .hex file with ST LINK Utility software
but I`m not able to use it with CubeIDE.Here is what I get when I press Debug Button:
Open On-Chip Debugger 0.10.0+dev-00021-g524e8c8 (2019-07-16-09:47)
```

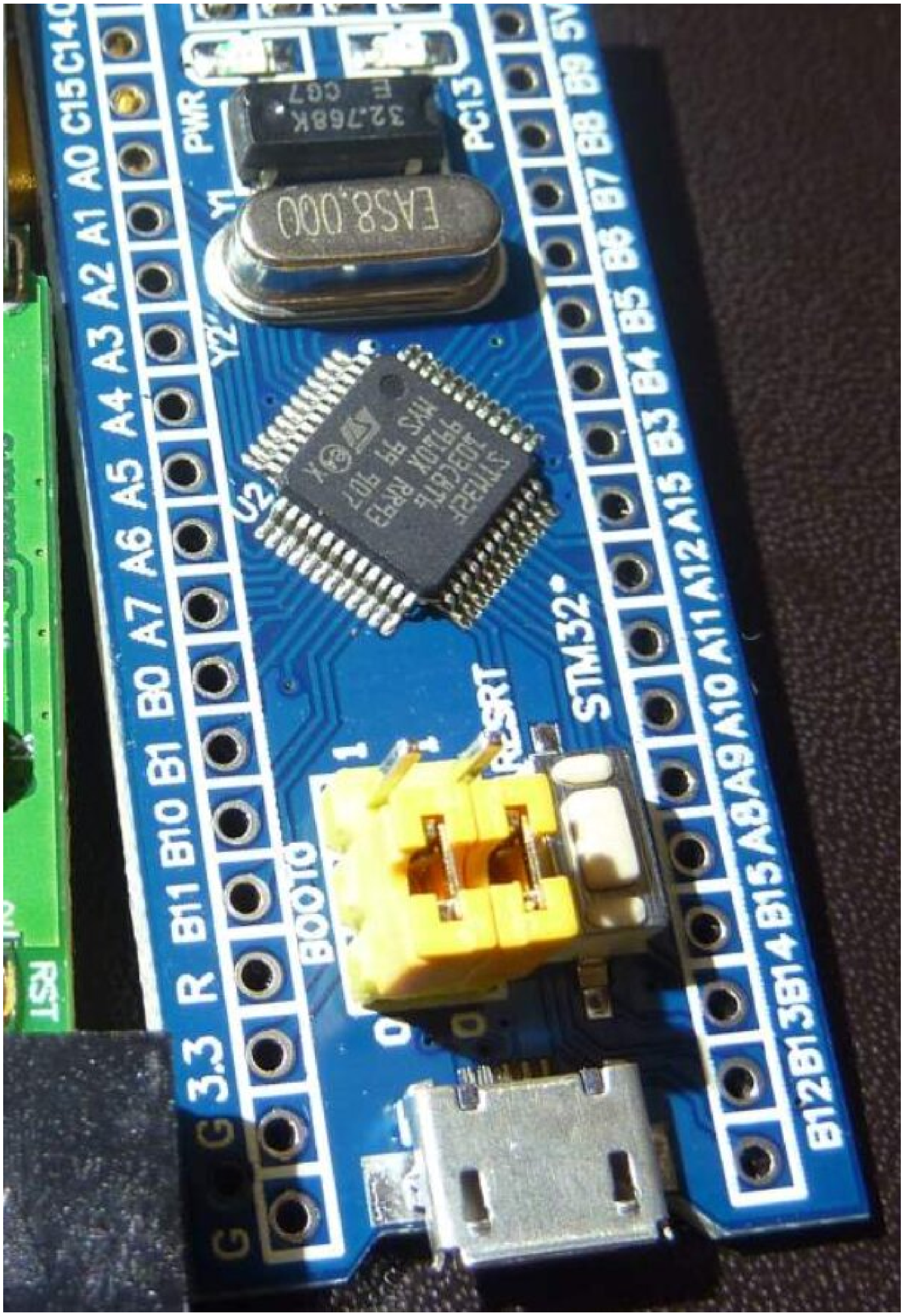
```
Licensed under GNU GPL v2
For bug reports, read
   http://openocd.org/doc/doxygen/bugs.html
none separate
Info : The selected transport took over low-level target
control. The results might differ compared to plain JTAG/SWD
adapter speed: 240 kHz
adapter_nsrst_delay: 100
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 240 kHz
Info : STLINK v2 JTAG v34 API v2 SWIM v7 VID 0x0483 PID
0x3748
Info : using stlink api v2
Info : Target voltage: 3.201053
Info : Stlink adapter speed set to 240 kHz
Warn : UNEXPECTED idcode: 0x2ba01477
Error: expected 1 of 1: 0x1ba01477
in procedure 'init'
in procedure 'ocd_bouncer'
I`ve found that, some solved that issue by pasting this line
to the some .cfg file on platformio IDE.
set _CPUTAPID 0x1ba01477
How could be this done on cubeIDE?
```

Blue Pill with a CKS32F103C8T6 chip

This chip is actually a CKS32F103C8T6, you can follow the backstory here: https://www.eevblog.com/forum/microcontrollers/stm32f103c8-diagnostic-bootable-binary-freely-available/msg2871198/#msg2871198

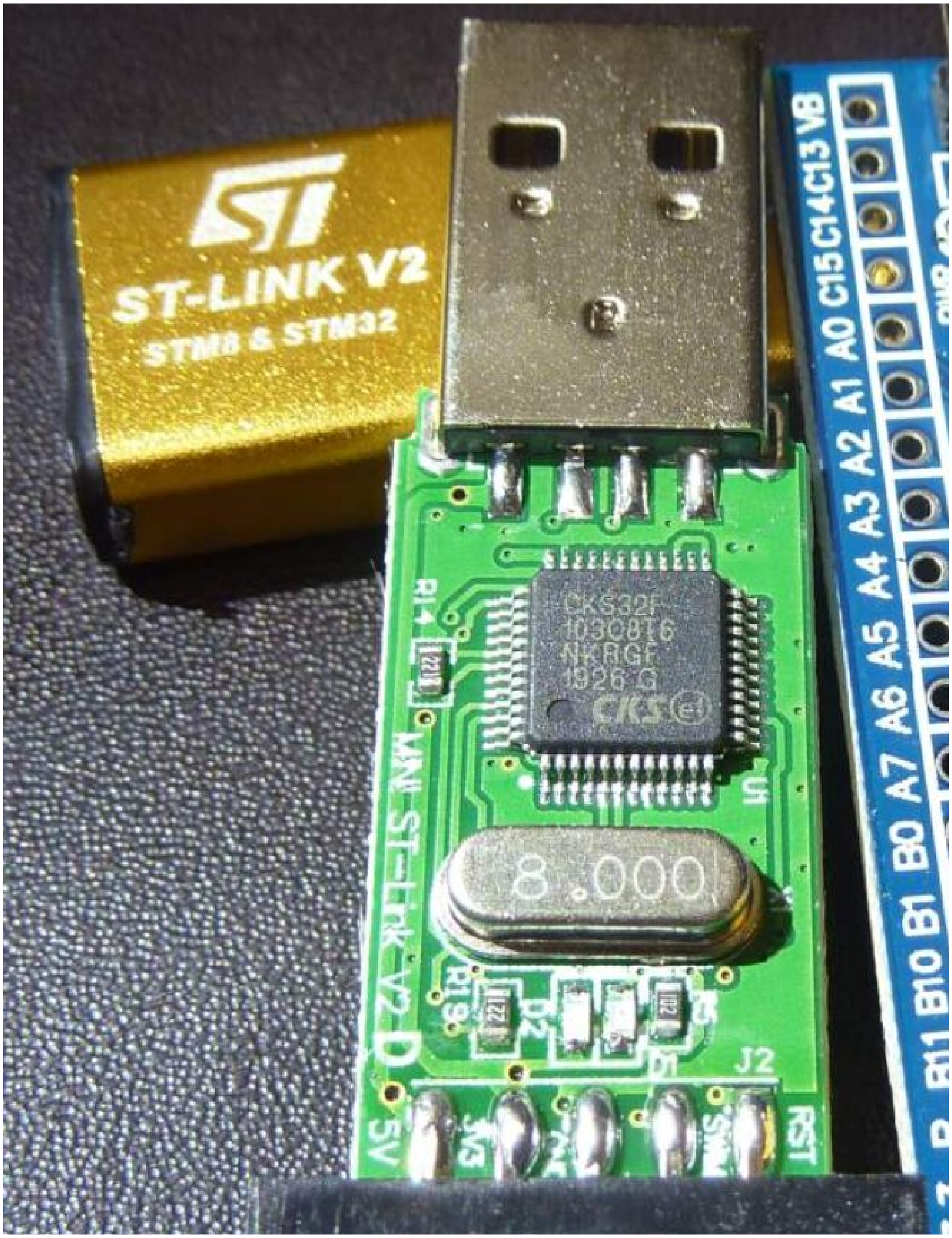
Blue Pill is like a box of chocolates

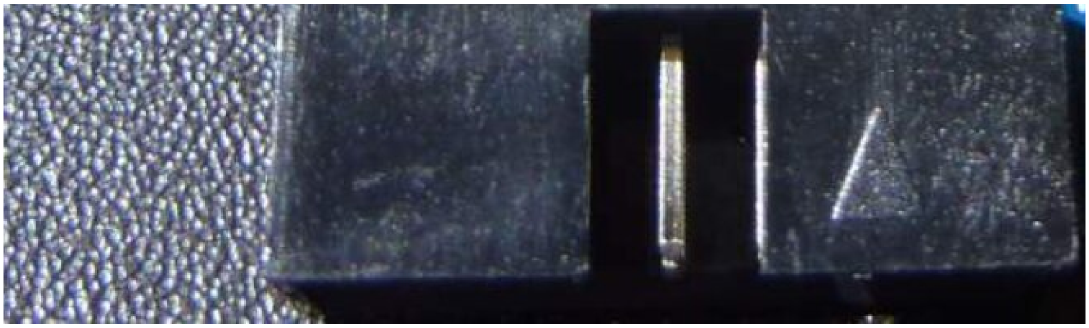



C14
C15
A0
A1
A2
A3
A4
A5
A6
A7
B0
B1
B10
B11
R
3.3
G
G
PWR
Y1
Y2
32.768K
EAS8.000
STM32F
103C8T6
U2
PC13
STM32
RESRT
BOOT0
1
0
5V
B9
B8
B7
B6
B5
B4
B3
A15
A12
A11
A10
A9
A8
B15
B14
B13
B12
RST

Then you have the Chinese ‘cheap USB to SWD programmers’ some of which don’t implement ST-Link properly. These also cost a couple of bucks and are usually bought with the ‘bargain’ $1.50 ‘Blue Pill’ because it is the only way to get in-chip debugging (usually used with GDB).

Update 2020: I see that some of the cheap Chinese ST-Link programmers now come with a CKS32F103C8T6 and update online with using the proper STM firmware!

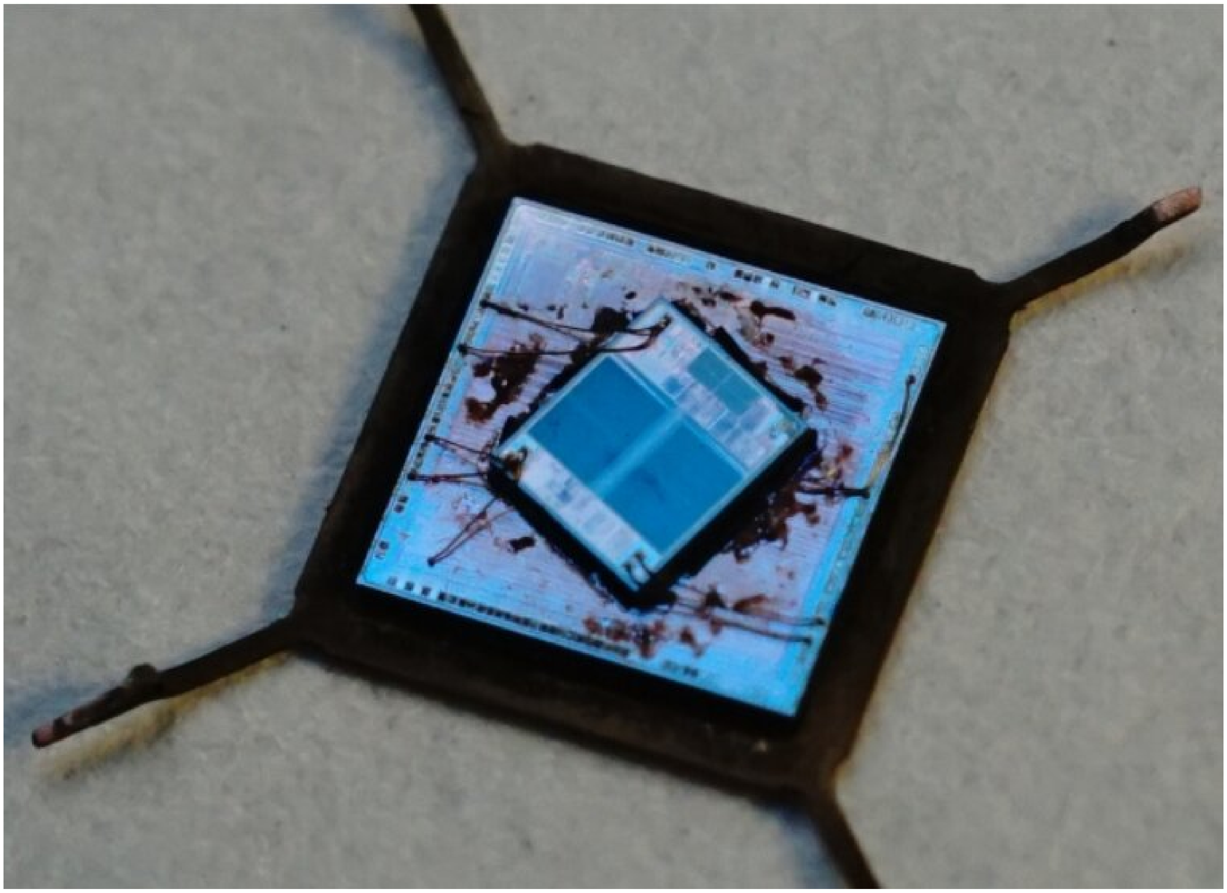

ST-LINK V2
STM8 & STM32
CKS32F
103C8T6
NKRGF
1926 G
CKS
8.000
MINI ST-Link V2
5V
3V3
SWIO
SWC
RST
J2
U1
C15 C14 C13 VB
A0 A1 A2 A3 A4 A5 A6 A7 B0 B1 B10 B11



GD32F103C8 Decapped

The GD32F103C8 is a bonafide STM32F103 “compatible” chip, it advertises its own ID, make and manufacturer identification internally. GD are a reputable and innovative manufacturer as far as I know.

GD32F103C8 Cortex-M3 die with serial flash; copied from https://zeptobars.com/en/read/GD32F103CBT6-mcm-serial-flash-Giga-Devices



Summary

I say cut the pain, make your time count and buy a new STM Nucleo or Discovery Board with the latest feature packed STM32M0, M0+ or M4 chip, a USB to SWD programmer/debugger is included for free, they aren't expensive and the smaller Cortex-M0+ can be had for $15. That way you will be learning and benefiting from the latest tech, not old tech.

And it's guaranteed to work, unlike that $1.50 'Blue Pill'.

Note

I have no connection with STM or any company, and receive no monies of any kind from any manufacturer or supplier, I use STM32 micros based on their ease of use and capability. If they were $20 a chip, I'd still consider them incredibly cheap.

Finally, some great articles on STM32 clones.

https://www.rogerclark.net/gd32f103-a-stm32f103-on-steroids/

https://olimex.wordpress.com/2015/11/09/chinese-clones-attack-stm32-microcontrollers/

REINVENTING THE WHEEL

I BUGS ME every time I read the comment **"why should we reinvent the wheel?"** usually by a non Forther commenting on the lack of Forth libraries and how they have tons of libraries for Arduino or Mbed or whatever. The implication is that these Libraries allow use of hardware that the C developer knows nothing about, thereby avoiding days or weeks studying datasheets until intimately familiar with the device.

OK, lets look at the WHEEL, who here has actually EVER made a REAL WORKING WHEEL ? … anyone ?

One big fault in logic is thinking "Mankind has invented the wheel, and I'm a part of Mankind, so I'm a part of that". No you're not, unless you have actually invented your own wheel, you don't have a clue, you only *think* you do!

My Dad once made a kid sized motorized Go-Cart and the wheels. The wheels were about a foot high and 8" in width. The wheel itself was rolled circular steel plate with a gap that accepted the ends of a 8" wide strip of tyre tread rubber which was tightly pulled around the rim before being locked in place in that gap. The wheel had welded steel spokes from rim to hub and each wheel could probably take a ton of weight. They were painted in silver paint.

I recall my Dad showing me the ‘secret’ to cutting the wide strips of rubber tread from bald tyres for the Go-Cart wheels, “the secret Terry”, he said “is to use a sharp knife and do it under *running water* and it cuts like butter”. And so it does.

The Go-Cart was a fantastic success with the small lawnmower engine that powered it. It had working brakes and did I mention it had a *engine* ? Many a glorious after school evening was spent in that machine as we kids did broadside after broadside at the end of our loose gravel driveway.

Until my Mother got involved …

She didn’t care about the broadsides, the dust we raised, the noise of the engine, the fuel we burned or the lack of seatbelts, this was around 1960 and Mum knew boys were tough, if we hurt ourselves she’d apply a bandage or calmly take us to the local doctor.

No it was *Dads wheels* that were the problem.

What she couldn’t abide were “home made wheels”, they were *unsightly* she claimed and berated my Father that her children deserved better. Mum didn’t have a clue about wheels, their load ratings, their strength and durability. She only knew wheels as a consumer who thinks prettier is better.

Dad, being a wise man knew my Mothers mind once set, could not be changed and dutifully purchased expensive pretty white smooth plastic wheels with inflatable tyres and replaced the wheels on the Go-Cart. They may have been wheelbarrow or trolley wheels I don’t know.

They lasted exactly one broadside, and produced a loud CLUNK. I was astounded to see a pretty new white wheel with inflatable tyre heading off down the street on its own. Yes, the plastic wheel had torn right off the hub which remained attached to the axle while the wheel and tyre made their dash for freedom.

That was the last time we drove that fantastic machine, Dad had blown his budget on the pretty plastic wheels (which were no good anyway) and Mum wouldn’t allow “Those Wheels” back on the Go-Cart. I pleaded and pleaded for the original wheels in vain.

Due to the psychological ramifications of this cruel and unusual child torture, when I grew up I became a motorcyclist instead of a car driver.

Two of Dads home made Go-Cart wheels went into a heavy electric arc welder cart and 50 years later are still rolling that welding machine along smoothly.

So Whats The Point Here ?

My Dad *really* knew how to make wheels because he invented some. Knowing my Dad invented wheels does not help me. To truly understand wheels I must invent some of my own.

- **Inventing the wheel is the best form of learning and the sooner you start the sooner you finish because there is no shortcut**

THE C PROGRAMMING LANGUAGE

When it comes to Embedded Electronics just ask Agent Smith what he thinks about C …

C is a plague, a cancer and FORTH is the CURE!!!

C is Bare Metal?

C users often claim that “C is low level, it’s at the bare metal” which is a interesting claim for a language that does *not support a “bit” type*. Compilers that do have it, use nonstandard extensions. You can use bitfields to pack flags, but they can produce rather inefficient code.

Ttmrichter Comments on the ‘C’ Language

> I absolutely despise C
>
> Basically, though, my stance is that C, whenever it faced a trade-off between writing code and maintaining code, chose the one that optimized the writing of code.
>
> They systematically chose, whenever there was a design choice point, to make the default behaviour “fuck maintenance”.

Which is why we have a language without arrays, say, just (lousy) syntax sugar that sorta/kinda approximates arrays among a myriad of other such flaws.

But hey! We can copy strings from one place to another with “while (*dest++ = *src++);”! As long as we don’t care about security! Or accidental overlaps! Or memory order sequencing! Or a million other things that matter in the real world, but make for code that doesn’t look so spiffy and cool!

It’s also why we have a language without any meaningful sense of modularity. Why the language had to get littered with so many stupid things like the “volatile” keyword. Why we have a language that has no meaningful constants.

Online C examples are useless

One of the big advantages touted for C is “Information hiding”, claimed to be a powerful programming technique because it reduces complexity.

Unfortunately this means C code examples are all but useless for a Forth programmer, because we need *all* the information to understand what is going on, but no one ever bothers to list their C header files etc.

- Even STM use The C Language To explain how their hardware works as tho C is the language of hardware.

Example 1

https://community.st.com/s/question/0D50X00009XkgyKSAR/stm32f0-sleep-mode “My following program blinks the LED using systick timer, while after ten times blink the LED the MCU should goes to sleep mode”

Note

Sleep mode is trivial in Forth, but I’m mystified where the WFI command is hiding in this example C code.

```
#include 'stm32f0xx.h'
#include 'stm32f0xx_tim.h'
#include <stm32f0xx_rcc.h>
volatile uint32_t msTicks;

void gpio_init(){

 GPIO_InitTypeDef GPIO_InitStructure;
 RCC_AHBPeriphClockCmd (RCC_AHBPeriph_GPIOC |
RCC_AHBPeriph_GPIOA,ENABLE);
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 |  GPIO_Pin_8 ;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
 GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
 GPIO_Init(GPIOC, &GPIO_InitStructure);

}

void blink_LED(){
 if (msTicks == 1000){GPIOC->ODR ^= GPIO_Pin_8; msTicks = 0;}

}

void SysTick_Handler (void) {
 msTicks++; // Increment counter
}
int main(){
 SysTick_Config (SystemCoreClock/1000); // Configure the
SYSTICK
 gpio_init();
 while(1){blink_LED();}
}
```

As a Forth user living in a C World, even the title of the example annoys me. It’s not a *program* it’s a *‘C’ program*.

To learn about STM32F0 Low Power Modes from this ‘example C program’ I need *all* the source, which is not provided or linked, for the files listed below :-

- stm32f0xx.h
- stm32f0xx_tim.h
- the full details of various *_InitStructure*

These kind of issues mean it’s rarely worth my time decoding a C “example”, tracking down the header details etc. It’s faster to go directly to the STM datasheets and programming reference manuals and just experiment in real time on the hardware itself.

Note

As an aside, I often observe that online C “examples” are just copy and paste of someone elses example code, complete with the same errors. I believe that the C Programming Language is so popular in Embedded because thousands of free libraries promise to do what the programmer wants whilst alleviating the need to actually understand the process.

C Users

Some C users are just not interested in learning. One *never* sees this kind of stupidity amongst Forth users.

Example

In December 2019 I created a free self bootable binary image with a menu for testing the STM32F103C8 MCU and announced it on https://www.eevblog.com/forum/microcontrollers/stm32f103c8-diagnostic-bootable-binary-freely-available/msg2830060/#msg2830060

All anyone has to do is flash the image to a STM32F103C8 and from a serial terminal run the menu. A child could do it, but not Doctorandus_P apparently.

```
Date: 08.01.2020
From: Doctorandus_P
Url:
https://www.eevblog.com/forum/microcontrollers/stm32f103c8-
diagnostic-bootable-binary-freely-
available/msg2861870/#msg2861870
Message:

I did not spend much time on this, as I'm not interested in
binary blobs, but still had a look...

Then I saw it was soe FORTH thing, and that done it, closed
the website.
I don't have anything against FORTH, but I am simply not
interested in learning yet another language.
```

Note

There is NO OTHER stm32f103c8 diagnostic bootable binary image available to my knowledge (@ Jan 2020), certainly nothing produced by any C user and made freely available.

Fortunately there are plenty of smart, polite and helpful C users on that same forum. In fact it was bingo600 who told me about the stm32f103c8 “hidden flash”.

```
Date: December 15, 2019
From: bingo600
Url:
https://www.eevblog.com/forum/microcontrollers/stm32f103c8-
diagnostic-bootable-binary-freely-
available/msg2830812/#msg2830812
Message:

@techman-001
Nice test  :-+

... Now to avoid those bluepill's on Ali/*bay

Thnx

/Bingo
```

THE C LIBRARY MYTH

Lets be perfectly clear here: Libraries in C++ (and in C of course) are just a joke. There’s no such thing as a library actually. They are just piles of definitions and functions/methods stacked together.

Those non-Forthers who think it’s not necessary to read every page of a STM32xxx manual because they have “libraries” that will do the job instead are really just strapping on pretty plastic wheels which will snap off at some point.

Why ? Because they don’t know what’s in the library. If they can’t be bothered reading a technical manual, why do you think they will be bothered scrutinising every line of someone else’s monster complex library code ?

Even a “low end” STM32F051 contains 37 Peripherals, 413 Registers and 3044 Register Bitfields, do you really think that anyone is going to write a Library that encompasses every permutation, for free, this century ?

Note

Mbed is my poster child for the Great Library Myth in this article, but it could just as easily be Arduino or Atollic Studio.

Nucleo-STM32L072 on MBED

Recently I used Mbed with a Nucleo-STM32L072. It’s probably the easiest method ever to get a board running as nothing is actually installed on the host PC other than the completed project binary downloaded from your Mbed cloud page and flashed to the Board MCU when it’s Dragged and Dropped into the Board bulk storage USB filesystem.

It has a number of code projects available for download which also make extensive use of libraries.

https://www.mbed.com/

Forth

I selected the traditional *Blinky* project and the Board led was flashing in 30 seconds … easy peasy. Sadly I didn’t have any idea about the actual blinky, how was the wait() function implemented, blocking spinner or interrupt ? What GPIO does myled() actually drive ?

However my biggest WTF! was the binary size of 26,076 Bytes. I can fit a whole interactive Forth OS in 19,000 bytes, and a Forth blinky in another 82 bytes.

That’s 6994 Bytes LESS !

Assembly

I can make the same Blinky in NON-INTERACTIVE Assembly code (see below) in 76 bytes. It’s cold bootable and completely self contained.

Thats 26,000 bytes less than the NON-INTERACTIVE MBED code and 19006 Bytes less than my INTERACTIVE Forth code.

Mbed Blinky Code

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1; // LED is ON
        wait(10.0); // 10 sec
        myled = 0; // LED is OFF
        wait(10.0); // 10 sec
    }
}
```

Binary file size = 26,076 Bytes.

Forth Blinky Code

```
\ The led is connected to GPIO Port C, bit 9
: half-second-delay 400000 0 do loop ;
: green-led.on   %1  9 lshift GPIOC_BSRR bis! ;
: green-led.off  %1  9 lshift GPIOC_BRR bis! ;
: blink
  %01  18 lshift GPIOC_MODER bis!       \ Make GPIO Port C,
bit 9 a OUTPUT
     do
        green-led.on
        half-second-delay
        green-led.off
        half-second-delay
     loop
;
```

Binary file size = 82 bytes

Assembly Blinky Code

```
// Minimal assembler blinky example for stm32f051
// Requires svd2as
// Board: stm32f0 Discovery
// Blue LED is on GPIOC bit 8
// 8th June 2018  Copyright <terry@tjporter.com.au> Released
under the GPL

.cpu cortex-m0                                    // Tell the
assembler what model of CortexM this is for
.thumb                                            // Cortex
micros only understand thumb(2) code
.text                                             // what
follows is code
.include "bitposn-equates.s"                      // a list
of ".equ  BIT0,    0x00000001", ".equ  BIT1," etc
.include "STM32F0xx.svd.s"                        //
Compliments of svd2gas
.syntax unified // use newer style instructions
Vector_Table:  .word     0x20000000               // stack
pointer value when stack is empty 0x20000000
ResetVector:    .word     start + 1               // Reset
Handler

start:
        ldr r1, = RCC_AHBENR
        ldr r2, = RCC_IOPCEN                      // Turn on
clock for GPIOC
        str r2, [r1]

moder:
        ldr r1, = GPIOC_MODER
        movs r2, 0b01                             // Set
GPIOC-8 to output
        lsls r2, GPIOC_MODER8_Shift               // BitWidth
```

```
2
        str r2, [r1]
                                                  // Main
Loop
led_on:
        ldr r1, = GPIOC_ODR
        ldr r2, = BLUE_LED_ON                     // Turn LED
on
        str r2, [r1]
        ldr r0, = DELAY                           // delay
delay_1: subs r0,r0,#1
        bne delay_1

led_off:
        ldr r1, = GPIOC_ODR
        ldr r2, = BLUE_LED_OFF                    // Turn LED
off
        str r2, [r1]
        ldr r0, = DELAY                           // delay
delay_2: subs r0,r0,#1
        bne delay_2

        b led_on                                  // branch
main_loop

constants:
        .equ DELAY, 0xfffff
        .equ BLUE_LED_ON, GPIOC_ODR8
        .equ BLUE_LED_OFF, 0x0
```

Binary file size = 76 Bytes.

Size of Binary Files

OS	Binary file size (bytes)	Notes	My Trust Rating (0-10) ?

OS	**Binary file size (bytes)**	**Notes**	**My Trust Rating (0-10) ?**
Mbed	26,076	spinner delay, non blocking, interrupt ? I don’t know, the doc doesn’t say	0.5
Forth	entire Forth OS = 19kb + 82 bytes	blocking spinner delay	10
Assembly	76	blocking spinner delay	10

USE MICROSOFT WINDOWS OR DIE

Systems
Devices
Premium Logo
Conficker Ready!
CERTIFIED FOR Conficker hosting
Basic Logo
Conficker Hosting BASIC
Works with Conficker MS™

It’s always BUGS ME when I read statements like “you have to interface with Windows or die” because when I was doing datacomms in 1985 the entire corporate world was IBM AIX, Silicon Graphics IRIX, Sun Microsystems SunOS, Hewlett-Packard HP-UX, DEC VMS etc … Universities were all Unix.

I remember seeing Windows 3.0 one day at work and all the engineers thought it was a joke, some child’s toy. Like *MSDOS* (upon which it depended) it didn’t even have native networking which came via a TSR that used the CMU TCP/IP stack because Bill Gates apparently thought that the Internet wasn’t useful.

I’ve always seen *MS Windows* as *White Goods Software* for the masses where “next years version” will have the egg timer you’ve all been waiting for.

I’ve watched *MS Windows* slowly spread like a cancer, driven not by any technical prowess but by cheap pricing, shark like salesmen and illegal market practices. *MS Windows* did bring some unique features to the corporate marketplace eventually however, people who took Unix yearly uptimes for granted were introduced to Windows BSOD’s, and Unix security was replaced by Windows only Viruses and Trojans.

Sure *MSDOS* was cheap and enabled me to run PADS and ORCAD at work, both very good programs and very fast, even on the glacially slow PC’s we had back then. Before *MSDOS* one needed a massively expensive Unix server and software for enterprise multilayer PCB CAD design, because that was *all there was*. Tektronix used to sell complete systems for PCB CAD, I remember seeing one at the local Tektronix office and I was mesmerized by the capability and the unbelievable cost.

What do you think the GE turbine design and CAD storage systems ran in the early 80’s, what did engineers use to fab all the TTL chips, Z80’s, 6502’s ? It sure wasn’t *MSDOS* or Windows. Autodesk’s Maya software is one of the most popular 3D animation packages in the history of the craft but it was made by Silicon Graphics and ran on Unix before Autodesk acquired it. Now before you go on about Autocad, in 1990, General Dynamics Electric Boat Corp chose CATIA as its main 3D CAD tool to design the U.S. Navy’s Virginia class submarine, and CATIA which originally ran on mainframes, was moved to Unix in 1988. CATIA didn’t run on Windows NT until 1998.

Linux did have a big effect on the small one man hitech business I owned and ran for 14 years. I did schematic capture, pcb cad, 2d cad, document processing, accounting, video surveillance and everything else I needed without any kind of *MS Windows*, but the main effect that Linux had was the competitive advantage it gave me over my competition. I did things the competition couldn’t imagine and couldn’t afford and I did it FAST with all the tools they didn’t have.

Sometimes I actually felt guilty, why should my life be so easy and theirs so hard ? While I was writing Perl and Sh scripts to do automated configuration and testing of WiFi P-P links, they were still trying to work out why their Windows XP laptops needed a reboot after a IP change.

Why did IBM recently pay $35 BILLION dollar$ for Red Hat Linux ? With that kind of money why didn’t they buy the awesome Windows 10 instead ;-)

Now I know the Windows diehards here will be swallowing their tongues, turning blue and harrumphing all over the place by now, but at 64 years of age, I could care less, and leave you with the following observations.

There is no such thing as Windows admin, there is only the *Windows Bandaid Kit* that the more advanced Windows users apply to the festering sores that grow on a OS they can never really understand while referring to the “Windows advanced book of 3001 Secrets” they have to use for a manual.

Real admin is only possible for Unix, most commonly Linux, and it takes TEN YEARS of daily use to become a competent Unix Admin, there is no shortcut, the sooner you start, the sooner you will arrive.

The Water Barons

Written 8th June 2019

A water baron is a person or organization that manages to gain dominant control over the water supply in an area

I’m very lucky, I was born in the 1950’s, a post war time of peace, at least in Australia. My Father and Uncles who were all in the Army survived and so I benefited from their skills and experiences as I grew.

Get Them When They’re Young

Around the age of nine I became obsessed with electronics after a kid at school gave me a pristine edition of the 1954 “ARRL Handbook”. I would spend the rest of my life learning, experimenting and working with electronics as a result. Thank you Johhny Coghlan.

Kids Are Always Broke

As a nine year old I didn’t have any money to fund purchases of cool MCU development boards from Amazon or

AliExpress, in fact none of those things existed back then and everything electronic used “Valves” because although Brattain, Bardeen and Shockley of Bell Labs had invented the transistor some 20 years before I was born, they were not yet found in common household devices like radios and televisions.

The 1954 “ARRL Handbook” was all about using Valves, making 300 - 10,000 volt power supplies, constructing the chassis, wiring, soldering, loom forming, coil winding, everything a young electronics enthusiast needed to obtain valuable hands on electronics skills. But reading is one thing, doing is quite another.

How Designers Are Made

Back then the local council *Rubbish Tip* “Tip” was my *only* source for electronics parts and I would eagerly accompany my Father when he disposed of household rubbish. I would search thru all the Tip refuse looking for radios or television sets or anything with electronic components. Later in my early teens I made a hand trolley and on weekends would walk the 5 miles to the Tip towing the trolley, search for cast out electronics and head home with anything useful.

I would then strip parts I thought useful, return the rest to my trolley where they would go back to the Tip on my next visit. I felt lucky to have this free resource nearby and who knew what may turn up there ? Over the years I found a CRT tube from a scope or radar set, a box full of Germanium transistors all of which were faulty <doh> and lots of valves, capacitors, resistors, transformers etc all salvaged from radios or tv’s.

I made a lot of gear using Tip parts and along the way I learnt how to design around what I had, to make what I wanted. In other words if the ARRl Handbook called for a 12AT7 valve and I only had a salvaged 12AU7, I'd have to design around that.

Building equipment this way taught me a lot about design. Compared to buying the right gear from a Internet based vendor it was much harder and delayed all my projects as I experimented to try and make things work, but I learnt a lot even if I didn't realise it at the time. Every single project was a journey into the unknown with many failures and few hard won successes.

We can learn as much from our failures as our successes.

By the time I was fifteen years old I could walk into any business and get a job as a electronics technician the same day. All I had to do was take a project or two with me that demonstrated my design, chassis building, soldering, wiring and looming skills. I'm not exaggerating when I say some employers were actually stunned when they saw my work because the school of "ARRL Handbook" made such good students ;-)

They Call it Stealing Now

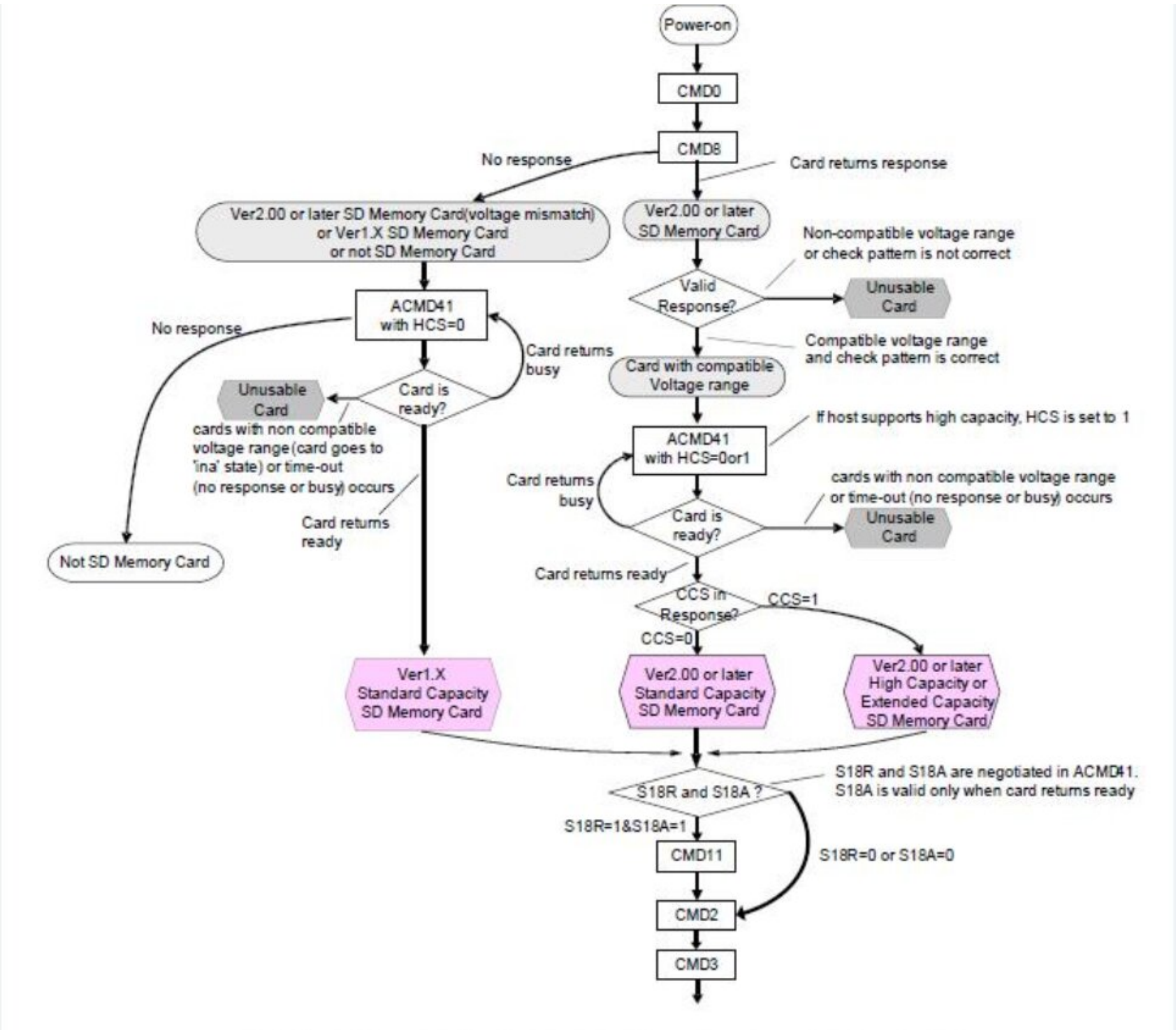
At some point the *money hungry arseholes*, running the council Tips realised that they could make money selling rubbish so up went barbed wire fences, locked gates and warning signs and my free and open access plus the access of EVERY GENERATION OF KIDS after me was abruptly terminated.

It still BUGS ME to see that kids nowadays have no access to rubbish tips like I had and that this opportunity is closed off forever for them.

THE HARD WAY

January 2020

Flowchart



Power-on
CMD0
CMD8
No response
Card returns response
Ver2.00 or later SD Memory Card(voltage mismatch) or Ver1.X SD Memory Card or not SD Memory Card
Ver2.00 or later SD Memory Card
Non-compatible voltage range or check pattern is not correct
Valid Response?
Unusable Card
Compatible voltage range and check pattern is correct
Card with compatible Voltage range
ACMD41 with HCS=0
No response
Card returns busy
Card is ready?
Unusable Card
cards with non compatible voltage range (card goes to 'ina' state) or time-out (no response or busy) occurs
Card returns ready
Not SD Memory Card
If host supports high capacity, HCS is set to 1
ACMD41 with HCS=0or1
Card returns busy
cards with non compatible voltage range or time-out (no response or busy) occurs
Card is ready?
Unusable Card
Card returns ready
CCS in Response?
CCS=1
CCS=0
Ver1.X Standard Capacity SD Memory Card
Ver2.00 or later Standard Capacity SD Memory Card
Ver2.00 or later High Capacity or Extended Capacity SD Memory Card
S18R and S18A are negotiated in ACMD41. S18A is valid only when card returns ready
S18R and S18A ?
S18R=1&S18A=1
S18R=0 or S18A=0
CMD11
CMD2
CMD3

Figure 4-2: Card Initialization and Identification Flow (SD mode)

Which option below best describes your initial reaction to this flowchart ?

1. Awesome, so that’s all I need to do to use a SD card ?
2. Ugh, can I have a library instead please ?

3. Honestly, I don’t have the time to read this stuff
4. It’s crap, I wrote a SD card library the other day in Forth check out *my* cool flowchart.

If you answered 4. then you definitely don’t need this site and I salute you. If you answered 1. welcome to this site I hope you find it interesting and helpful.

Choices 2. and/or 3. mean this site isn’t for you, don’t waste another second with embedded electronics, leave immediately and good luck in some *other* area that does interest and motivate you.

The Old

When I taught myself electronics I only had one book, the 1954 ARRL handbook, and various scrap electronics hardware I collected from the local rubbish tip. No one I knew had any electronics experience or interest and the only person who did, the local “radio and tv repair shop” was a one man band. As a result the guy was very busy and seemed annoyed when I tried to ask him electronics questions from the front counter of his shop in my local town, so after a couple of visits, with great disappointment, I realised that avenue was a dead end.

Back in those days, around 1966 there were no home computers or Internet in Australia. Hobby electronics was relegated to Ham Radio as far as I know, and “Hams” built all their own gear.

Amazingly I managed to survive and flourish using my own efforts which leads me to this rant.

The New

I read this post online today :

Today, I posted my first (Stack Exchange) question trying to be very specific and described my troubleshooting efforts leading up to the post. It got a response about 2 minutes later claiming it was a duplicate and already answered in another thread. When I went to that thread, it was only slightly related and completely unhelpful - but my thread was now closed. Is this normal?

An experienced poster replied:

It is typical, unfortunately. There are a lot of “reputation” gamers, who post answers and close questions as duplicates even when they’re not (and often also close questions because they themselves do not understand the question, especially when the question is hard). They do not really do it to help others, but to get more “reputation”. The entire site is based on the idea that “best answers get voted to the top”, but popularity and quality do not correlate much (and causate even less).

The Bad and The Worse

Today in 2020, it seems to me that the bulk of people learning electronics and programming naturally turn to the Internet. Sadly many Internet resources (apart from PDF component specs and manufacturer component data) are worse than useless.

Worse Than Useless?

Bad information is worse than no information, and the Internet of 2020 is jam packed with *incorrect* information, *disinformation* and information *tainted* by vested interests such as the Stack Exchange example above and vendors casting doubts on reliability/safety of their competitions products.

Internet learning in 2020 is a minefield and the question must be asked, is it a minefield worth the risk to cross ?

Keep It Simple

If you want to learn real embedded electronics, start with something simple and understand *every step* of the process by yourself and keep building, one step after another.

In general (apart from datasheets), forget the Internet for learning Embedded Electronics, it’s mainly a sales tool now, run by the best marketing types on Earth.

Work it out yourself, you already have the most awesome resource available, the human brain!

Note

simple doesn’t mean easy, quick, or cheap.

Asking For Help

My philosophy is “never, ever ask until you have absolutely nothing left to try, until every avenue of investigation has been examined and you are totally exhausted and running on empty”. Learning is like lifting weights, the harder it is, the better the results and the bigger ones *brain* muscle grows.

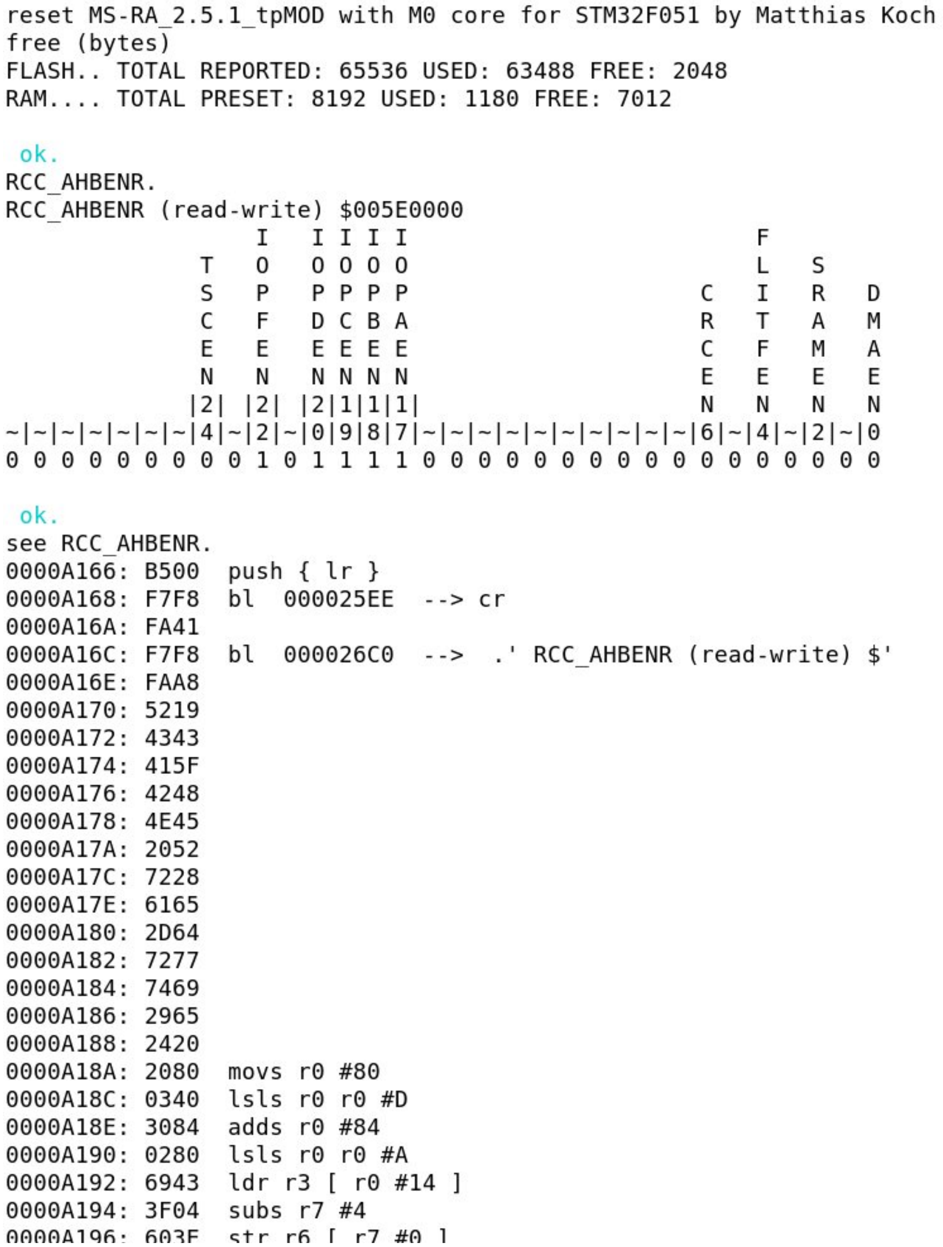
A Learning Challenge

This is a simple $5 handmade Forth embedded computer, build one ?

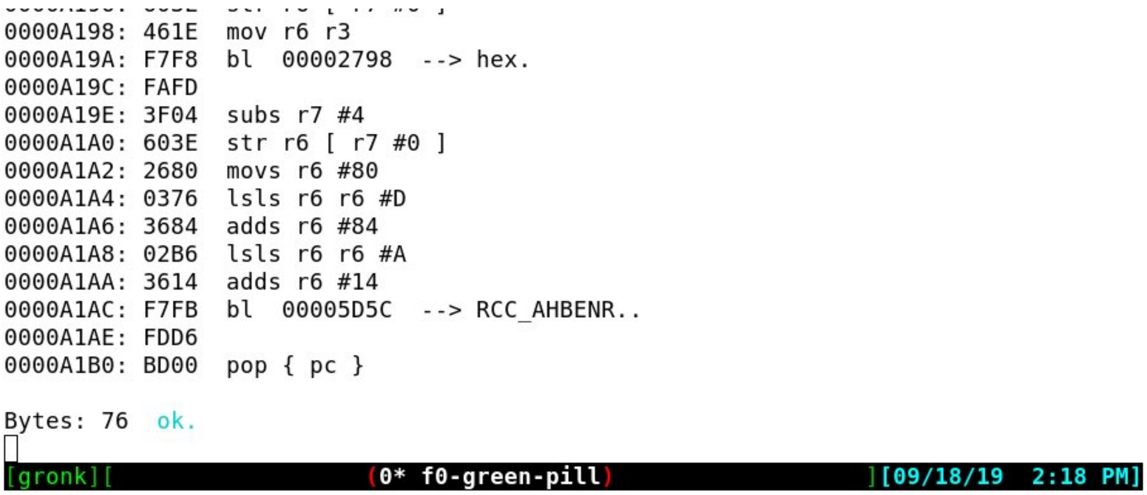
Note

Everything you need is on this site.

Terminal view

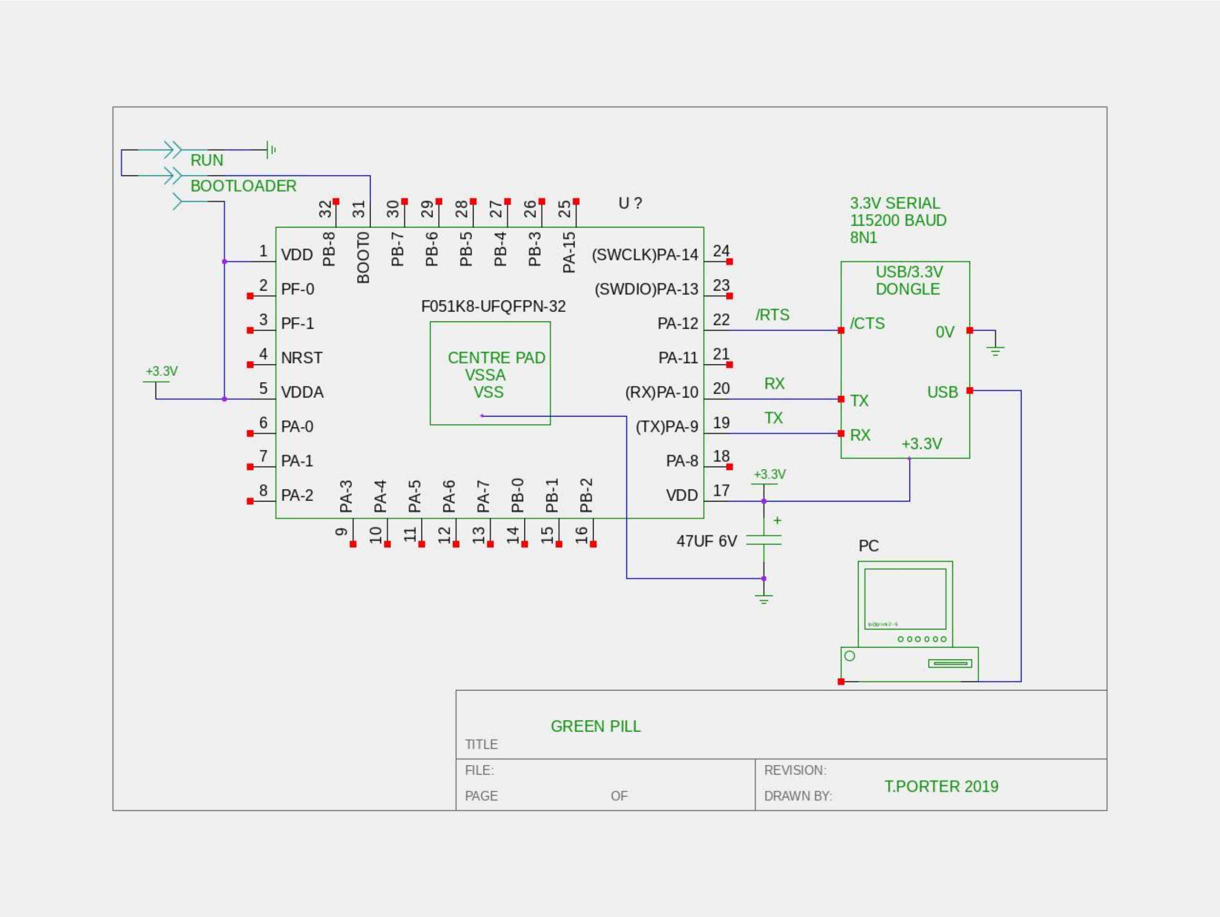


```
reset MS-RA_2.5.1_tpMOD with M0 core for STM32F051 by Matthias Koch
free (bytes)
FLASH.. TOTAL REPORTED: 65536 USED: 63488 FREE: 2048
RAM.... TOTAL PRESET: 8192 USED: 1180 FREE: 7012

 ok.
RCC_AHBENR.
RCC_AHBENR (read-write) $005E0000
                  I   I I I I                         F
              T   O   O O O O                         L   S
              S   P   P P P P                     C   I   R   D
              C   F   D C B A                     R   T   A   M
              E   E   E E E E                     C   F   M   A
              N   N   N N N N                     E   E   E   E
             |2| |2| |2|1|1|1|                    N   N   N   N
~|~|~|~|~|~|~|4|~|2|~|0|9|8|7|~|~|~|~|~|~|~|~|~|~|6|~|4|~|2|~|0
0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
see RCC_AHBENR.
0000A166: B500  push { lr }
0000A168: F7F8  bl  000025EE  --> cr
0000A16A: FA41
0000A16C: F7F8  bl  000026C0  -->  .' RCC_AHBENR (read-write) $'
0000A16E: FAA8
0000A170: 5219
0000A172: 4343
0000A174: 415F
0000A176: 4248
0000A178: 4E45
0000A17A: 2052
0000A17C: 7228
0000A17E: 6165
0000A180: 2D64
0000A182: 7277
0000A184: 7469
0000A186: 2965
0000A188: 2420
0000A18A: 2080  movs r0 #80
0000A18C: 0340  lsls r0 r0 #D
0000A18E: 3084  adds r0 #84
0000A190: 0280  lsls r0 r0 #A
0000A192: 6943  ldr r3 [ r0 #14 ]
0000A194: 3F04  subs r7 #4
0000A196: 603E  str r6 [ r7 #0 ]
```



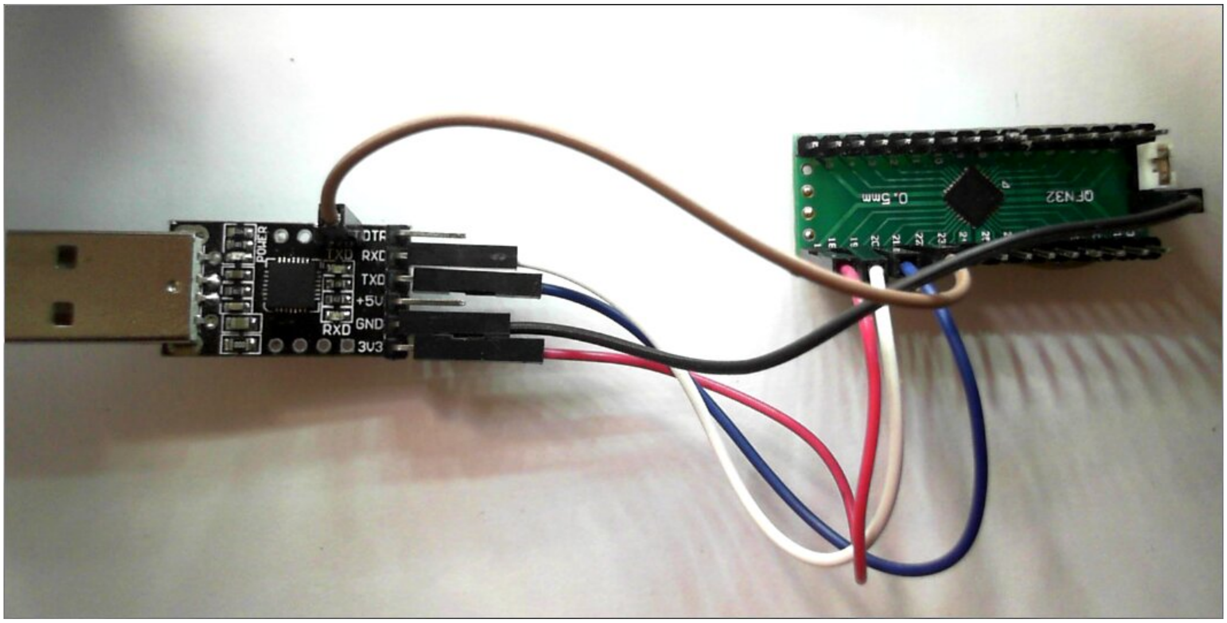
```
0000A198: 461E  mov r6 r3
0000A19A: F7F8  bl  00002798  --> hex.
0000A19C: FAFD
0000A19E: 3F04  subs r7 #4
0000A1A0: 603E  str r6 [ r7 #0 ]
0000A1A2: 2680  movs r6 #80
0000A1A4: 0376  lsls r6 r6 #D
0000A1A6: 3684  adds r6 #84
0000A1A8: 02B6  lsls r6 r6 #A
0000A1AA: 3614  adds r6 #14
0000A1AC: F7FB  bl  00005D5C  --> RCC_AHBENR..
0000A1AE: FDD6
0000A1B0: BD00  pop { pc }

Bytes: 76  ok.

[gronk][                    (0* f0-green-pill)                    ][09/18/19  2:18 PM]
```

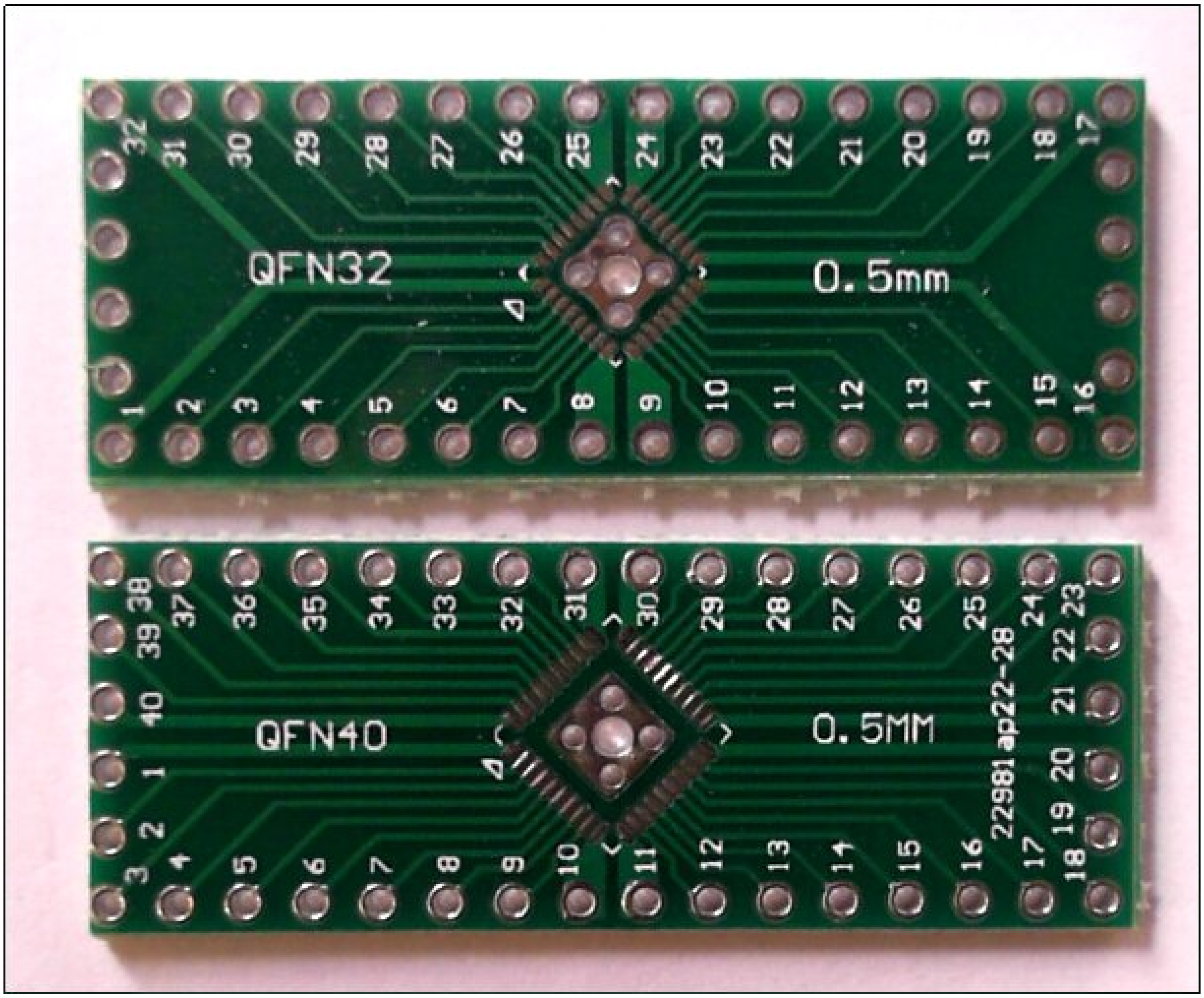
Schematic

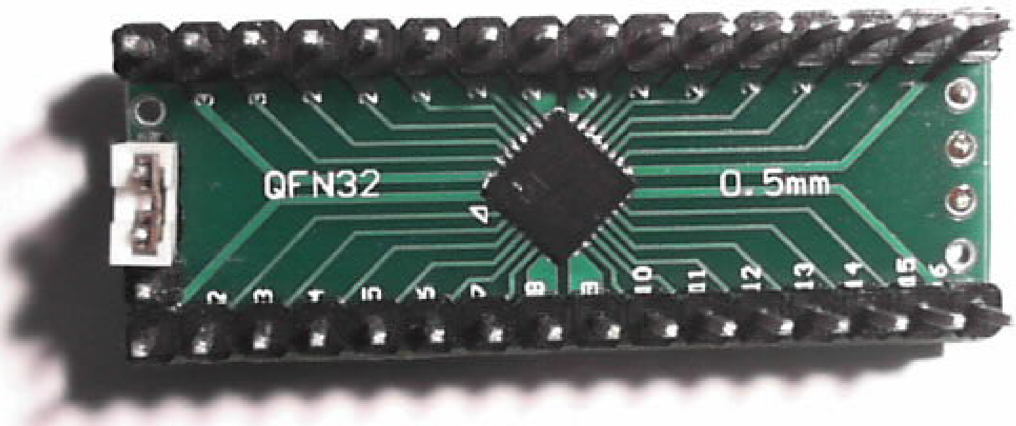


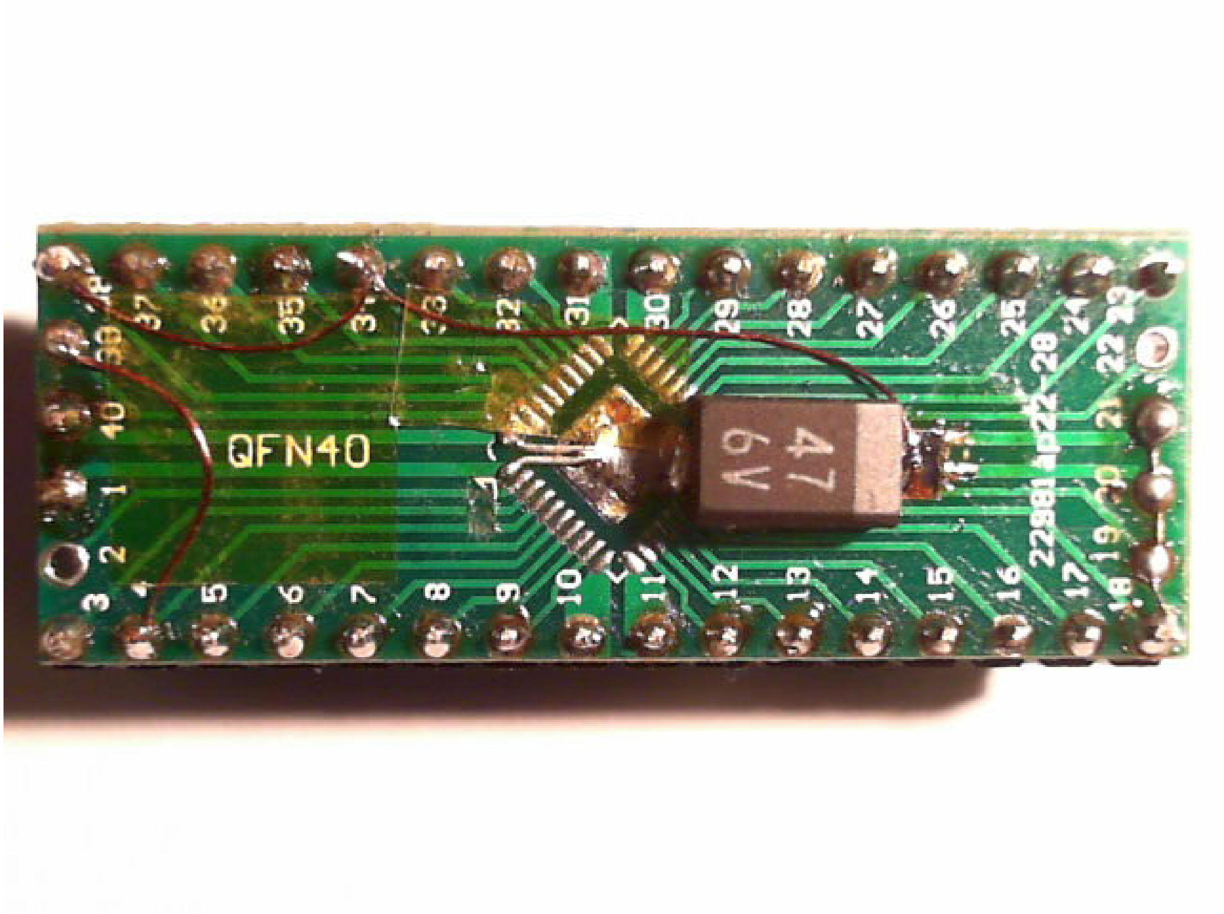
RUN
BOOTLOADER
U ?
F051K8-UFQFPN-32
VDD
PF-0
PF-1
NRST
VDDA
PA-0
PA-1
PA-2
PB-8
BOOT0
PB-7
PB-6
PB-5
PB-4
PB-3
PA-15
PA-3
PA-4
PA-5
PA-6
PA-7
PB-0
PB-1
PB-2
(SWCLK)PA-14
(SWDIO)PA-13
PA-12
PA-11
(RX)PA-10
(TX)PA-9
PA-8
VDD
CENTRE PAD
VSSA
VSS
+3.3V
47UF 6V
3.3V SERIAL
115200 BAUD
8N1
USB/3.3V
DONGLE
/RTS
/CTS
0V
RX
TX
USB
+3.3V
PC
TITLE
GREEN PILL
FILE:
PAGE
OF
REVISION:
DRAWN BY:
T.PORTER 2019

Construction


POWER
TXD
RXD
DTR
RXD
TXD
+5V
GND
3V3
QFN32
0.5mm


QFN32
0.5mm
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
QFN40
0.5MM
22981ap22-28
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40


QFN32
0.5mm
10
11
12
13
14
15



Hackerspaces or SalesPlaces?

25 Jan 2020

A hackerspace is a community-operated, often “not for profit”, workspace where people with common interests, such as computers, machining, technology, science, digital art, or electronic art, can meet, socialize and collaborate –Wikipedia

Hackaday.com

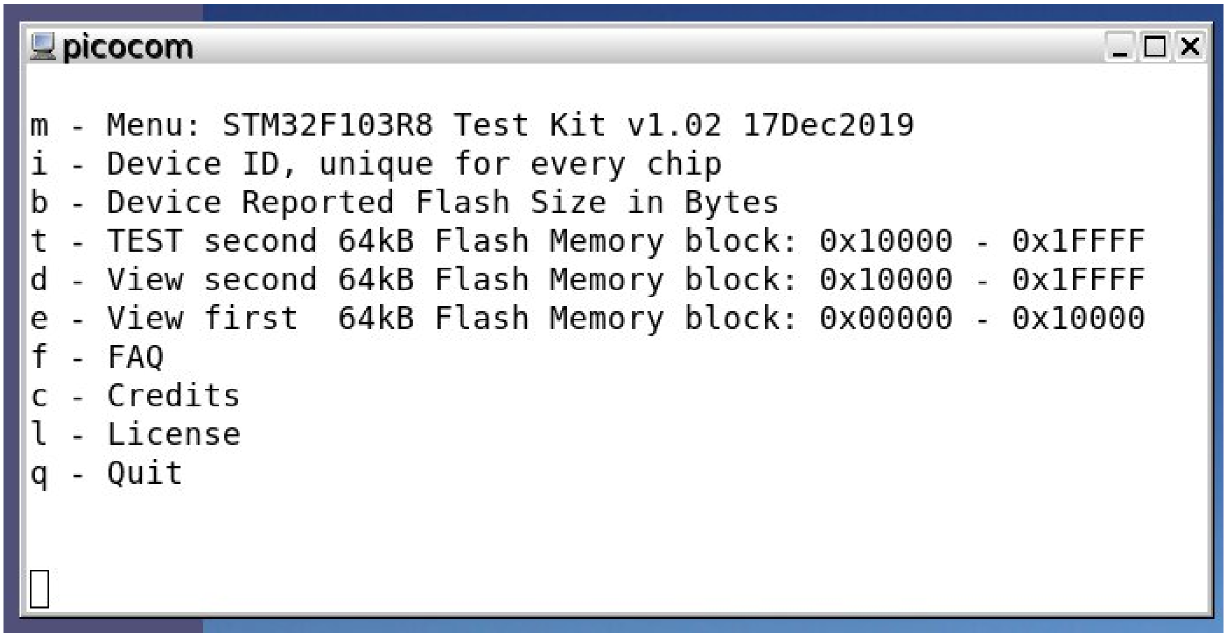
https://hackaday.com/ is an online pseudo “hackerspace” business which describes itself like so:

Hackaday serves up Fresh Hacks Every Day from around the Internet. Our playful posts are the gold-standard in entertainment for engineers and engineering enthusiasts.

Personally I think Hackaday is more of a pewter standard than a gold one, because to me the Hackaday shine has lost its lustre and the thin gold plating wears off a little bit more every time they pretended an advertisement is an interesting article about technology.

One thing that bugs me is that it seems Forth is *persona non grata* at Hackaday.

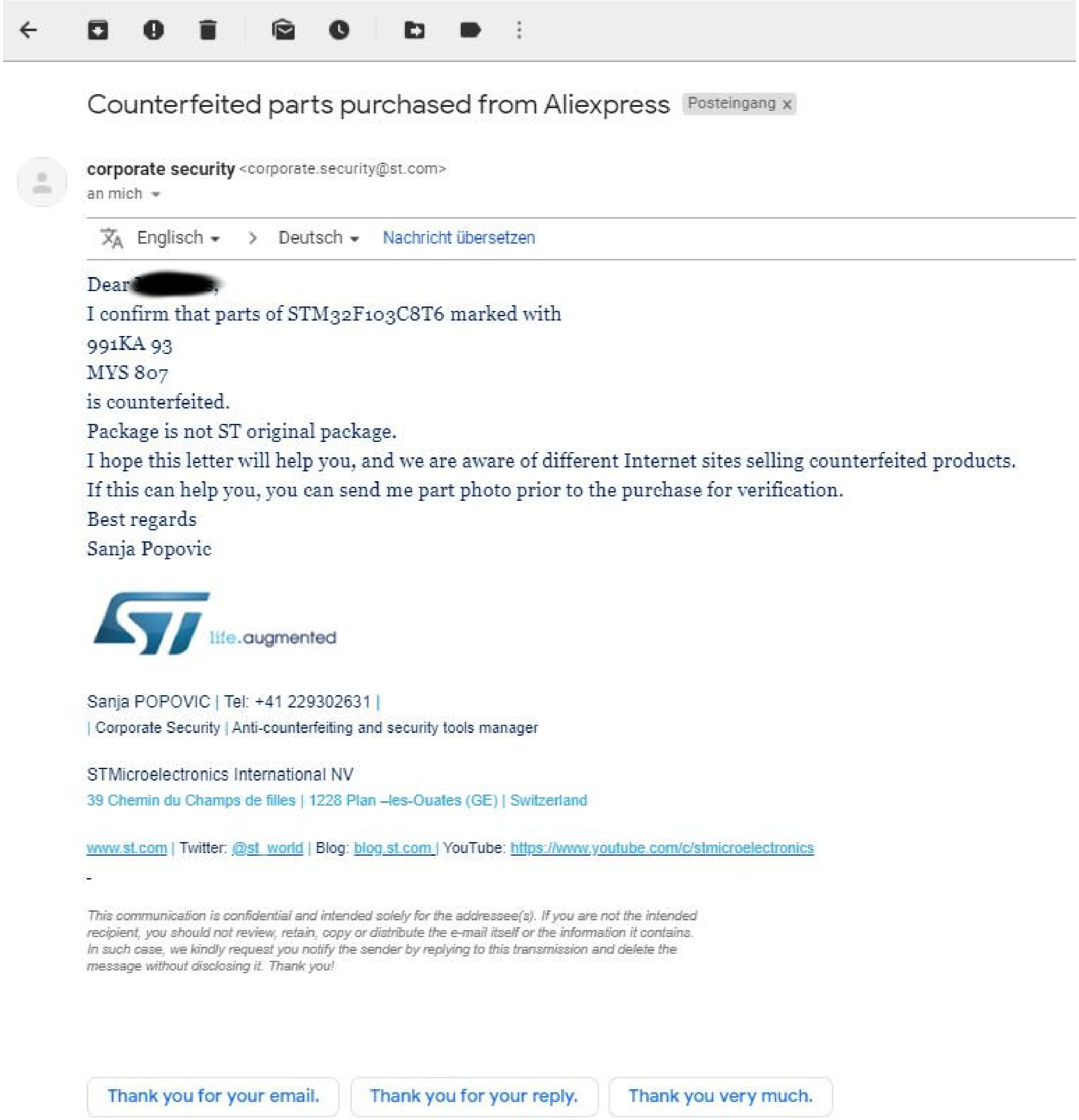
- Hackaday last posted a Forth article exactly 12 months ago on January 25, 2019 titled “pocket-forth-invades-your-ti-calculator” https://hackaday.com/2019/01/25/pocket-forth-invades-your-ti-calculator/
- They last wrote about Mecrisp-Stellaris Forth in 2017, 3 years ago.
- I submitted my STM32F103C8 Diagnostics project to Hackaday on Dec 17, 2019 and I’ve heard nothing from them. I think it’s a useful project and *nothing* like it exists to the best of my knowledge. With this program, Blue Pill owners can determine if they have a bona fide STM32F103 chip or a counterfeit chip and whether it has a hidden extra 64kB of flash memory. Given there are millions of Blue Pill units in circulation, I think this is worthy of publication ? The free binary has been downloaded about twice a day in the last month, mainly by Windows users in the USA. No Forth knowledge is required and it’s operated from a simple menu.



```
m - Menu: STM32F103R8 Test Kit v1.02 17Dec2019
i - Device ID, unique for every chip
b - Device Reported Flash Size in Bytes
t - TEST second 64kB Flash Memory block: 0x10000 - 0x1FFFF
d - View second 64kB Flash Memory block: 0x10000 - 0x1FFFF
e - View first  64kB Flash Memory block: 0x00000 - 0x10000
f - FAQ
c - Credits
l - License
q - Quit
```

Counterfeit Chips

This is one reason why the STM32F103C8 Diagnostics project is useful, helping to detect FAKE chips.



Counterfeited parts purchased from Aliexpress

corporate security <corporate.security@st.com>
an mich

Dear [redacted],
I confirm that parts of STM32F103C8T6 marked with
991KA 93
MYS 807
is counterfeited.
Package is not ST original package.
I hope this letter will help you, and we are aware of different Internet sites selling counterfeited products.
If this can help you, you can send me part photo prior to the purchase for verification.
Best regards
Sanja Popovic

life.augmented

Sanja POPOVIC | Tel: +41 229302631 |
| Corporate Security | Anti-counterfeiting and security tools manager

STMicroelectronics International NV
39 Chemin du Champs de filles | 1228 Plan –les-Ouates (GE) | Switzerland

www.st.com | Twitter: @st_world | Blog: blog.st.com | YouTube: https://www.youtube.com/c/stmicroelectronics

-

This communication is confidential and intended solely for the addressee(s). If you are not the intended recipient, you should not review, retain, copy or distribute the e-mail itself or the information it contains. In such case, we kindly request you notify the sender by replying to this transmission and delete the message without disclosing it. Thank you!

A real Beauty?

On January 21, 2020 Hackaday published a article titled “CircuitPython Slithers Into 100th Board — The OHS 2020 Badge”.

This article builds up the hype about this ‘cool’ *badge*, remarking that **you can’t actually buy one** because the badge won’t be for sale, but if you purchase an admission to the *OSHW Summit 2020* ($30 – $190) “you will be handed one of these beauties”.

Beauty is in the Eye of the Beholder

“Circuit Python” is a project by AdaFruit, it only runs on expensive, complex, vastly capable MCU’s such as Cortex-M4 with a minimum 256kB flash and plenty of Ram. Overall it’s about 30x slower than Forth on the same MCU, except Forth will run nicely on a cheap, highly capable Cortex-M0 class MCU with 32kB Flash and 4kB Ram, which Circuit Python will **never be able to do**.

Hackaday reader “Cyk” comments:

> I stopped using Circuitpython when I needed interrupts and timers, and had to learn that Circuitpython simply doesn’t have them. So yeah, maybe a decent learner’s toy, but as so often a dead end, as soon as you advance.

Cyk could do far more with a $2 Blue Pill than the crippled Circuit Python powered ‘OHS 2020 Badge’

Today’s Gold Standards

Todays gold-standard in entertainment for engineers and engineering enthusiasts at Hackaday include the following:

1. A Raspberry Pi Terminal That’s Always Within Reach
2. Students Use Low Tech Hacks On High Tech Parking Enforcer
3. Pi 4 Emulator In A Durable, Dumpstered Cabinet
4. A Simple Science Fair AM Transmitter

5. A Pair Of CRTs Drive This Virtual Reality Headset
6. BinDayCator Lets You Know When To Take Out The Trash
7. Spring Clamp Is Completely 3D Printed

5. looks interesting, the rest not so much.

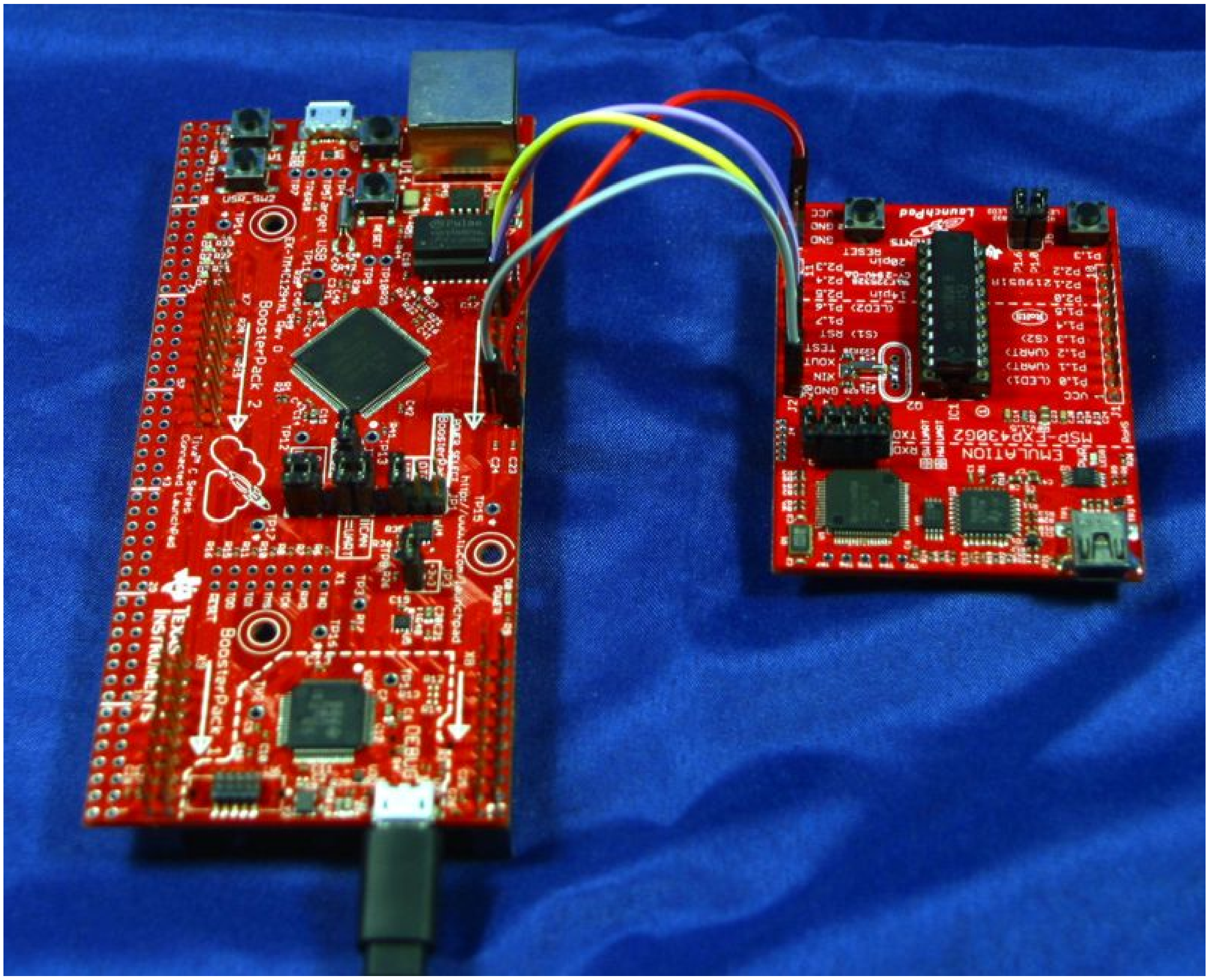
Perhaps Hackaday haven’t published my STM32F103C8 Diagnostics project because they’re still trying to work out how to do it with the C programming language now they are aware of the idea ;-)

This isn’t the only submission that silently vanished into the Hackaday *black hole*.

Tethered Forth

Nov 6, 2018 I emailed Hackaday about a brilliant Tethered Forth system Matthias has created, named *Mecrisp-Across*. I was so impressed I created a separate documentation site for it at https://mecrisp-across-folkdoc.sourceforge.io/

Apart from a reply email informing me the reviewer was travelling and would get back to me, I never heard from Hackaday again.



Tethered Forths provide the compact binary generation of The C Language **PLUS** the interactivity of Forth, *all in the one system*.

If that doesn't excite you, WTF are you doing here ?

Is no one at Hackaday able to understand the implications of such a system, do they consider their readers too dumb to understand it, or is it just a "Forth doesn't sell" attitude ?

You'll have to decide :)

Projects: Working Forth Programs

Note

All my projects are licensed under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later versions see http://www.gnu.org/licenses/. . Projects by other contributors are covered by the licence terms in their code.

Note

All project code is CMSIS-SVD compliant for easy code reuse.

- You will need to load memory map words for all registers used. See Svd2forth-v2 for a easy way to generate a memmory mapped file for any of the projects below.

Window Comparator

- Hardware: STM32F0 Discovery Board using a STM32F0xx MCU, but any STM32F0 with a Comparator should work
- Clock: 8 Mhz using the stm32f0 target chip internal RC clock as default with Mecrisp-Stellaris

- This program demonstrates Comparator 1 & 2 in ‘Window’ mode by lighting the BLUE and GREEN LEDS depending on voltage (potentiometer position), and gives a colored voltage status on the terminal.
- standalone, no other files needed

DOWNLOAD: `f051-comparator-tests.fs`

A Sunrise and Sunset Calculator

By Andrew Palm. Requires his maths library be loaded first.

DOWNLOAD: `sunrise-sunset.fs`

Note

Usage:

1. set-zenith-code.

zenith code	type of sunrise/sunset
0	Official sunrise and sunset
1	civil twilight
2	nautical twilight
3	astronomical calculations

2. Run “sunrise-sunset-utc” with inputs lat, long, year, month, day.
3. To see the local sunrise and sunset times, plus hours of daylight and local solar noon, use the word “show-local-times” with input the integer utc offset for the time zone.

Examples

```
#include sunrise-sunset-v2.fs

0 set-zenith-code
-33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
11 show-local-times

1 set-zenith-code
-33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
11 show-local-times

2 set-zenith-code
-33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
11 show-local-times

3 set-zenith-code
-33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
11 show-local-times

\ 0 set-zenith-code  ok.
\ -33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
\  ok.
\ 11 show-local-times
\ Sunrise:        06:41:07
\ Sunset:         19:39:14
\ Daylight:       12:58:07
\ Solar noon:     13:10:10
\  ok.
\
\ 1 set-zenith-code  ok.
\ -33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
\  ok.
\ 11 show-local-times
\ Sunrise:        06:15:32
\ Sunset:         20:04:47
\ Daylight:       13:49:15
\ Solar noon:     13:10:10
\  ok.
\
\ 2 set-zenith-code  ok.
\ -33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
\  ok.
\ 11 show-local-times
\ Sunrise:        05:45:13
\ Sunset:         20:35:04
\ Daylight:       14:49:50
\ Solar noon:     13:10:08
\  ok.
\
\ 3 set-zenith-code  ok.
\ -33,750000 150,700000  2018 02 25  sunrise-sunset-utc cr
\  ok.
\ 11 show-local-times
\ Sunrise:        05:13:59
\ Sunset:         21:06:15
\ Daylight:       15:52:16
\ Solar noon:     13:10:07
\  ok.

Australian Metrological published times, Penrith, Sydney,
Australia for 2018 02 25
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
\ 06:16 EDT   first light
\ 06:41 EDT   sunrise
\ 19:39 EDT   sunset
\ 20:05 EDT   last light
```

See also

[s31.32 Insights]

Benchmark Your Forth MCU ?

- Hardware: STM32F0 Discovery Board using a STM32F0xx MCU
- Clock: 8 Mhz
- This program uses a *Greatest Common Divisor* algorithm to benchmark CPU speed, and measures the time with a Interrupt driven Sysclock. A list of other system results is included in the source file.
- Instructions are in the sourcecode

`Download: Benchmark`

Push button switch debounce demo

- Hardware: STM32F0 Discovery Board using a STM32F0xx MCU
- Clock: 8 Mhz using the stm32f0 target chip internal RC clock as default with Mecrisp-Stellaris
- Must be used with the e4thcom Terminal. This is interactive, press return at each pause.
- This program demonstrates the effect of increasing the debounce delay while counting USER push button presses.
- The MCU just spins to provide the debounce delay

`Download: Push button switch debounce demo`

Square Wave Generator

- Hardware: STM32F0 Discovery Board using a STM32F0xx MCU
- Clock: 8 Mhz using the stm32f0 target chip internal RC clock as default with Mecrisp-Stellaris
- Duty Cycle: 50 %
- Frequency Range: 123 Hz to 350 KHz
- Waveform: Square wave at 3.0 Volts

- Output: on pin PB-1
- Registers required: RCC, GPIOB, TIM14
- Files required: freq-gen-memmap.txt and freq-gen.txt
- Upload to your F0 Mecrisp-Stellaris Forth MCU in this order: freq-gen-memmap.txt, freq-gen.txt and you should see a 5000 Hz square wave appear on PB-1
- Enter “<freq> hz” at the Forth prompt to change the frequency. “<freq>” must be between 123 and 350000 or PB-1 will just go to a High level.

`Download: freq-gen-memmap.txt` **`Download: freq-gen.txt`**

Screenshot:-

```
Forth> 20000 hz
-> 20000 hz
<- for a output of 20000 Hz (CLOCK = 8MHz, Prescaler = 0) TIM14_ARR is $00000190
<- ok.
```

100 Microsecond Systick

- STM32F0 Discovery Board using a STM32F0xx MCU
- Utilises the 8MHz MCO crystal controlled frequency from the SWD programmer
- Overclocked to 96 MHz using the crystal derived 8 MHz MCO clock from the SWD programmer.
- GPIOA_8 may be monitored to confirm the overclock is working. Look for 48MHz (1/2 clock) on this pin.
- A marker pulse on GPIOB_2 is generated every Systick for external measurement. Period = 100us, Width = 800 nS
- Flashes the BLUE LED (on GPIOC_8) at a rate of 1 second

`Download: 100 Microsecond Systick`

Overclock to 96 MHz

- STM32F0 Discovery Board using a STM32F0xx MCU
- Utilises the 8MHz MCO crystal controlled frequency from the SWD programmer
- Increases the default Mecrisp-Stellaris speed by 12 times
- Optional MCO frequency monitor pin on GPIOA-8
- Power usage STM32F051: 20mA (3mA @ 8MHz)
- May not program Flash memory reliably, but seems to run from Ram without any problems.

`Download: Overclock to 96 MHz`

F0 Discovery 48 MHz System Clock

`Download: f0-disco-48mhz.fs`

Delay using the interrupt driven 1 millisecond Systick

- STM32F0 Discovery Board using a STM32F0xx MCU
- Default Mecrisp-Stellaris 8Mhz RC system clock
- Blue LED blinks at one second Interval
- 1mS marker pulse is available on PB2 for scope fine tuning
- ‘msdelay’ word provides a accurate time delay via the systick
- By Terry Porter based on code by kfoltman

`Download: msdelay`

Timer, 1mS Interrupt Systick

- STM32F0 Discovery Board using a STM32F0xx MCU
- Default Mecrisp-Stellaris 8Mhz RC system clock
- Interrupt driven with 1mS Systick
- Blue LED blinks at one second Interval
- 1mS marker pulse is available on PB2 for scope fine tuning
- Use to time software or hardware. ‘delay’ example included
- Standalone code, no other files needed
- By Terry Porter based on code by kfoltman

`Download: timer.fs`

Analog To digital 19ch scanner

- STM32F0 Discovery Board using a STM32F0xx MCU
- Default Mecrisp-Stellaris 8Mhz RC system clock
- No DMA or interrupts used
- Scans all 19 available channels of the ADC: PA0-7, PB0-1, PC0-5, VREF, VBAT and Temperature sensor in Degrees C
- It works by scanning each channel and pushing the result on the Stack.
- The End Of Conversion Flag is checked by a spinner.
- After the sequence of 19 channel conversions is finished, it pops the results off the stack, scales and displays them along with the number of spins.
- By Terry Porter

`Download: Analog To digital 19ch scanner`

Morse code generator

- STM32F0 Discovery Board using a STM32F0xx MCU
- Default Mecrisp-Stellaris 8Mhz RC system clock
- LEDS blink for DASH and DOT’s
- PA4 drives a piezo loudspeaker to hear the Morse Code
- By quaak.haak, Terry Porter

`Download: Morse code generator`

Recognisers

What is a recogniser ?

A recognizer (not a compsci word) attempts to work out what to do when given a string or a token.

There are two types of string or token; words or literals (aka numbers). There are 3 things you can do with a word; take its XT and EXECUTE it, COMPILE, it or POSTPONE it. There are 2 things you can do with a (floating point) number; put it on the (float) stack, or (F)LITERAL it to put it on the (float) stack at run time.

A recognizer takes a token and tries to recognize it based on some characteristic. Can I find it in a dictionary? It’s word. Does it look like a floating point number? It’s a float. Etc.

If it recognizes the token, the recognizer may transform it into some usable value; like a number, or an NT (name token) and so on. It also returns a value (for instance, a pointer to a 3 element table of actions) that represents the kind of token it recognized, and that value tells the system what to do with the converted token when interpreting, compiling or postponing (for instance, selecting one of the 3 actions in the table and executing it).

The reason that it’s of so much interest is that it allows pluggable standardized recognizers for a variety of tokens. ANS Forth is readily extensible, except in this key area. [alexrivadpm]

Question: So the recognizer if successful is taking a token and tagging it with (value, action) and passing that back to forth to process.

Answer: Yes, that’s about the sum of it. [alexrivadpm]

References

[andrea.ferro] https://www.quora.com/Why-is-Java-always-said-to-have-a-garbage-collector-Doesnt-every-high-level-language-have-one

[andreas.wagner] Andreas maintains the MK20DX256 and CY8C5888 ports. Nick: pointfree on #freenode and reddit. My email is andreas.wagner@lowfatcomputing.org

[alexrivadpm] Subject: Re: bikeshedding, Newsgroups: comp.lang.forth, Date: Mon, 11 Dec 2017

[atollic.studio] https://www.st.com/en/development-tools/truestudio.html

[davespace] http://www.davespace.co.uk/arm/

[Dave-Wyland] https://blogs.mentor.com/colinwalls/blog/2010/10/19/go-forth/?cmpid=8688 (Commented on 20 October 2010 at 18:12)

[e.d.rather] Elithabeth D Rather http://forth.com who I consider to be one of the greatest Forth ambasadors ever!

[gefffox] Geff Fox http://www.ultratechnology.com/antiansi.htm

[gforth] https://www.gnu.org/software/gforth/

[hertaville] http://www.hertaville.com/external-interrupts-on-the-stm32f0.html

[manfredmahlow]

Manfred Mahlow https://wiki.forth-ev.de/doku.php/en:projects:e4thcom#e4thcom-061

[matthias.koch] m-atthias@users.sf.net. The author of Mecrisp, Mecrisp-Stellaris, Mecrisp-Across, Mecrisp-Ice, Mecrisp-Quintus

[mtrute] Matthias Trute http://amforth.sourceforge.net/

[Rasmus-Friis-Kjeldsen] Rasmus Friis Kjeldsen http://reblag.dk/stm32/

[reginald.beardsley] https://sourceforge.net/p/mecrisp/discussion/general

[ttmrichter] Known as the Nick “ttmrichter” on irc.freenode.net. https://github.com/github/linguist/pull/748#issuecomment-37377476

[vanderhorst] a.w.m.van.der.horst http://home.hccnet.nl/a.w.m.van.der.horst/forthlecture5.html

> What’s New ? <

CMSIS-SVD

Let’s assume you’re new to ARM Cortex-M MCU’s, you have a nice new board of some kind, perhaps a STM32 Discovery Board, or a Tiva Connected Launchpad etc and you have installed Mecrisp-Stellaris Forth by Mathias Koch https://sourceforge.net/projects/mecrisp/ on it.

Now you’d like to begin programming, but where do you start, the Cortex-M documentation is vast, and there are a LOT of Registers involved. Can you imagine the work involved in manually creating the Forth Words for Registers and Bitfields ?

Some statistics for various MCU’s that run Mecrisp or Mecrisp Stellaris.

MCU	Peripherals	Registers	Register Bitfields	Clock Max
MSP430G2553	17	113	503	
STM32F0xx	37	413	3044	45
STM32F303x	38	549	3857	
STM32F103xx	53	722	4833	75
TM4C129x	65	2137	7347	120
STM32F7x	89	1737	14091	
STM32F7x7	93	2093	17051	200

CMIS-SVD

ARM created CMSIS-SVD to automate the job of telling the industry ‘standard’ C programs about Cortex-M Peripherals, Registers and Register Bitfields with SVDConv.exe, however that is of no use to Forth users so svd2forth (and others) were develped, initially based on work by Ralph Doering as cmsis-svd-fth (https://github.com/ralfdoering/cmsis-svd-fth).

Depending on the size of your MCU Flash memory, it’s quite possible to install all the Register Memory Mapped Words in

Flash, so that Mecrisp-Stellaris knows about every Register, or you can just select the Memory Mapped Words you need.

CMIS-SVD files

- Are XML text files detailing all Registers (or most of them at least) for MCU families and not individual parts.
- May be downloaded from https://github.com/posborne/cmsis-svd/archive/master.zip (39MB) or found in some of the packages here for which svd files have been specially created.
- Programming language MCU specific files are created by applying XSLT stylesheets to the CMIS-SVD files.

Note

Do you use STM32xxxxx ? see below for svd2forth-v3-stm32.zip

Note

Do you use MSP430 ? Svd2mecrisp is available for MSP430xx and the SVD2forth for it is below.

Note

Do you use Tiva Connected Launchpad ? see below for a TM4C1294NCPDT svd2forth.

Svd2forth

- Was created to use CMSIS-SVD files for automated Mecrisp-Stellaris Forth memory-mapped Word production, making target software development easier and less error prone, especially for Forth beginners.
- Enforces CMSIS compliant naming across all Mecrisp-Stellaris programs, enabling code reuse.
- Developed on FreeBSD but also tested on a Ubuntu 16.04.1 x64 VM. This is for Unix and won't run on Microsoft Windows unless you install all the stuff that turns Windows into Unix.

Svd2forth Downloads

`Download: svd2forth-v1-1.0.zip` <– Original Bare Bones Minimal Version

`Download: svd2forth-v3-stm32.zip` <– SVD2FORTH with the LOT: Register Pretty Printing, bitfield.fs, memmap.fS and EQU.s files with SVD description fields

`Download: SVD2FORTH-V3-STM32F103C8T6.zip release` <– Special Version Just For The STM32F103C8T6 (Blue Pill)

Non STM32 svd2forth Downloads

`Download:svd2forth-TM4C1294NCPDT-1.1.zip`

`Download:msp430g2553-2forth-1.0.zip`

Latest STM32 Release

`Svd2forth-v3-stm32` uses a MCU CMSIS-SVD file to auto generate every peripheral and register memory mapped Word, and every register bitfield manipulation Word for the *ENTIRE* MCU. This should work on all STM32xxxxx.svd files.

It probably won’t work with non STM32 CortexM files which have slightly different SVG’s.

New: 22June2019, also produced is a STM32xxxxx.svd.equates.s is for use with the GNU ASsembler which may be handy if you’re doing any Mecrisp-Stellaris Assembly coding.

Features

The memmap.fs file contains all memory mapped peripheral Words and can print all register contents in a group or singly. The print Word for every register has a simple 32 bit legend showing the bit number, this applies to registers which only use 15 bits or different groupings. Both bitfields.fs and memmap.fs now contain description fields parsed from the SVD. Hopefully this will reduce the need to have a databook open while coding.

For instance “rcc.” will print every RCC peripheral register. However “RCC-IOPSMEN.” will just print the named register which is part of the RCC group of registers.

The bitfield.fs file contains a Word for each bitfield to be pasted in full or part as desired.

The peripherals contained in either file can be de-selected by commenting out the peripheral in template.xml as shown in the readme.

A STM32L07x.svd is included in the tarball for use with the Makefile.

See the **`svd2forth-v3-stm32.zip`** README for more information.

Screenshot Printing the RCC Peripheral Registers

For a STM32L07x MCU, taken from a Nucleo-STM32L073 running on my desk.

```
rcc.

rcc. RCC-CR () $00000305
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0
1

RCC-ICSCR () $0059B063
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0 1
1

RCC-CRRCR () $00003900
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0
0

RCC-CFGR () $00000005
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
1

RCC-CIER (read-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
```

```
RCC-CIFR (read-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-CICR (read-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-IOPRSTR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-AHBRSTR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-APB2RSTR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-APB1RSTR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-IOPENR (read-write) $0000008F
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
```

```
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1
1

RCC-AHBENR (read-write) $00000100
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0

RCC-APB2ENR (read-write) $00004000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-APB1ENR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-IOPSMEN (read-write) $0000009F
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1
1

RCC-AHBSMENR (read-write) $01111301
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0
1

RCC-APB2SMENR (read-write) $00405225
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0
1
```

```
RCC-APB1SMENR (read-write) $F8FE4A33
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1
| 0
1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 1
1

RCC-CCIPR (read-write) $00000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC-CSR () $1C000000
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|1|1|1|1|1|
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1
| 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

 ok.
```

Example RCC Bitfields

For a STM32L07x MCU

```
Note: these are meant to be hand picked as needed, not
uploaded to the MCU or the memory will soon be filled!
 Replace 'bis!' (set bit) with 'bic!' to CLEAR bit, 'bit@' to
test bit etc.

\ RCC-CR ()
: RCC-CR_PLLRDY   %1 25 lshift RCC-CR bis! ;  \ RCC-CR_PLLRDY
PLL clock ready flag
: RCC-CR_PLLON   %1 24 lshift RCC-CR bis! ;  \ RCC-CR_PLLON
PLL enable bit
: RCC-CR_RTCPRE   ( %XX -- ) 20 lshift RCC-CR bis! ;  \ RCC-
CR_RTCPRE    TC/LCD prescaler
: RCC-CR_CSSLSEON   %1 19 lshift RCC-CR bis! ;  \ RCC-
CR_CSSLSEON    Clock security system on HSE enable  bit
: RCC-CR_HSEBYP   %1 18 lshift RCC-CR bis! ;  \ RCC-CR_HSEBYP
HSE clock bypass bit
```

```
: RCC-CR_HSERDY   %1 17 lshift RCC-CR bis! ;  \ RCC-CR_HSERDY
HSE clock ready flag
: RCC-CR_HSEON   %1 16 lshift RCC-CR bis! ;  \ RCC-CR_HSEON
HSE clock enable bit
: RCC-CR_MSIRDY   %1 9 lshift RCC-CR bis! ;  \ RCC-CR_MSIRDY
MSI clock ready flag
: RCC-CR_MSION   %1 8 lshift RCC-CR bis! ;  \ RCC-CR_MSION
MSI clock enable bit
: RCC-CR_HSI16DIVF   %1 4 lshift RCC-CR bis! ;  \ RCC-
CR_HSI16DIVF    HSI16DIVF
: RCC-CR_HSI16DIVEN   %1 3 lshift RCC-CR bis! ;  \ RCC-
CR_HSI16DIVEN    HSI16DIVEN
: RCC-CR_HSI16RDYF   %1 2 lshift RCC-CR bis! ;  \ RCC-
CR_HSI16RDYF    Internal high-speed clock ready  flag
: RCC-CR_HSI16KERON   %1 1 lshift RCC-CR bis! ;  \ RCC-
CR_HSI16KERON    High-speed internal clock enable bit for
some IP kernels
: RCC-CR_HSI16ON   %1 0 lshift RCC-CR bis! ;  \ RCC-
CR_HSI16ON    16 MHz high-speed internal clock  enable

\ RCC-ICSCR ()
: RCC-ICSCR_MSITRIM   ( %XXXXXXXX -- ) 24 lshift RCC-ICSCR
bis! ;  \ RCC-ICSCR_MSITRIM    MSI clock trimming
: RCC-ICSCR_MSICAL   ( %XXXXXXXX -- ) 16 lshift RCC-ICSCR
bis! ;  \ RCC-ICSCR_MSICAL    MSI clock calibration
: RCC-ICSCR_MSIRANGE   ( %XXX -- ) 13 lshift RCC-ICSCR bis! ;
\ RCC-ICSCR_MSIRANGE    MSI clock ranges
: RCC-ICSCR_HSI16TRIM   ( %XXXXX -- ) 8 lshift RCC-ICSCR bis!
;  \ RCC-ICSCR_HSI16TRIM    High speed internal clock
trimming
: RCC-ICSCR_HSI16CAL   ( %XXXXXXXX -- ) 0 lshift RCC-ICSCR
bis! ;  \ RCC-ICSCR_HSI16CAL    nternal high speed clock
calibration

\ RCC-CRRCR ()
: RCC-CRRCR_HSI48CAL   ( %XXXXXXXX -- ) 8 lshift RCC-CRRCR
bis! ;  \ RCC-CRRCR_HSI48CAL    48 MHz HSI clock  calibration
: RCC-CRRCR_HSI48RDY   %1 1 lshift RCC-CRRCR bis! ;  \ RCC-
CRRCR_HSI48RDY    48MHz HSI clock ready flag
: RCC-CRRCR_HSI48ON   %1 0 lshift RCC-CRRCR bis! ;  \ RCC-
CRRCR_HSI48ON    48MHz HSI clock enable bit

\ RCC-CFGR ()
: RCC-CFGR_MCOPRE   ( %XXX -- ) 28 lshift RCC-CFGR bis! ;  \
RCC-CFGR_MCOPRE    Microcontroller clock output  prescaler
```

```
: RCC-CFGR_MCOSEL   ( %XXX -- ) 24 lshift RCC-CFGR bis! ;  \
RCC-CFGR_MCOSEL    Microcontroller clock output  selection
: RCC-CFGR_PLLDIV   ( %XX -- ) 22 lshift RCC-CFGR bis! ;  \
RCC-CFGR_PLLDIV    PLL output division
: RCC-CFGR_PLLMUL   ( %XXXX -- ) 18 lshift RCC-CFGR bis! ;  \
RCC-CFGR_PLLMUL    PLL multiplication factor
: RCC-CFGR_PLLSRC   %1 16 lshift RCC-CFGR bis! ;  \ RCC-
CFGR_PLLSRC    PLL entry clock source
: RCC-CFGR_STOPWUCK   %1 15 lshift RCC-CFGR bis! ;  \ RCC-
CFGR_STOPWUCK    Wake-up from stop clock  selection
: RCC-CFGR_PPRE2   ( %XXX -- ) 11 lshift RCC-CFGR bis! ;  \
RCC-CFGR_PPRE2    APB high-speed prescaler  APB2
: RCC-CFGR_PPRE1   ( %XXX -- ) 8 lshift RCC-CFGR bis! ;  \
RCC-CFGR_PPRE1    APB low-speed prescaler  APB1
: RCC-CFGR_HPRE   ( %XXXX -- ) 4 lshift RCC-CFGR bis! ;  \
RCC-CFGR_HPRE    AHB prescaler
: RCC-CFGR_SWS   ( %XX -- ) 2 lshift RCC-CFGR bis! ;  \ RCC-
CFGR_SWS    System clock switch status
: RCC-CFGR_SW   ( %XX -- ) 0 lshift RCC-CFGR bis! ;  \ RCC-
CFGR_SW    System clock switch

\ RCC-CIER (read-only)
: RCC-CIER_CSSLSE   %1 7 lshift RCC-CIER bis! ;  \ RCC-
CIER_CSSLSE    LSE CSS interrupt flag
: RCC-CIER_HSI48RDYIE   %1 6 lshift RCC-CIER bis! ;  \ RCC-
CIER_HSI48RDYIE    HSI48 ready interrupt flag
: RCC-CIER_MSIRDYIE   %1 5 lshift RCC-CIER bis! ;  \ RCC-
CIER_MSIRDYIE    MSI ready interrupt flag
: RCC-CIER_PLLRDYIE   %1 4 lshift RCC-CIER bis! ;  \ RCC-
CIER_PLLRDYIE    PLL ready interrupt flag
: RCC-CIER_HSERDYIE   %1 3 lshift RCC-CIER bis! ;  \ RCC-
CIER_HSERDYIE    HSE ready interrupt flag
: RCC-CIER_HSI16RDYIE   %1 2 lshift RCC-CIER bis! ;  \ RCC-
CIER_HSI16RDYIE    HSI16 ready interrupt flag
: RCC-CIER_LSERDYIE   %1 1 lshift RCC-CIER bis! ;  \ RCC-
CIER_LSERDYIE    LSE ready interrupt flag
: RCC-CIER_LSIRDYIE   %1 0 lshift RCC-CIER bis! ;  \ RCC-
CIER_LSIRDYIE    LSI ready interrupt flag

\ RCC-CIFR (read-only)
: RCC-CIFR_CSSHSEF   %1 8 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_CSSHSEF    Clock Security System Interrupt  flag
: RCC-CIFR_CSSLSEF   %1 7 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_CSSLSEF    LSE Clock Security System Interrupt  flag
: RCC-CIFR_HSI48RDYF   %1 6 lshift RCC-CIFR bis! ;  \ RCC-
```

```
CIFR_HSI48RDYF    HSI48 ready interrupt flag
: RCC-CIFR_MSIRDYF   %1 5 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_MSIRDYF    MSI ready interrupt flag
: RCC-CIFR_PLLRDYF   %1 4 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_PLLRDYF    PLL ready interrupt flag
: RCC-CIFR_HSERDYF   %1 3 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_HSERDYF    HSE ready interrupt flag
: RCC-CIFR_HSI16RDYF   %1 2 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_HSI16RDYF    HSI16 ready interrupt flag
: RCC-CIFR_LSERDYF   %1 1 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_LSERDYF    LSE ready interrupt flag
: RCC-CIFR_LSIRDYF   %1 0 lshift RCC-CIFR bis! ;  \ RCC-
CIFR_LSIRDYF    LSI ready interrupt flag

\ RCC-CICR (read-only)
: RCC-CICR_CSSHSEC   %1 8 lshift RCC-CICR bis! ;  \ RCC-
CICR_CSSHSEC    Clock Security System Interrupt  clear
: RCC-CICR_CSSLSEC   %1 7 lshift RCC-CICR bis! ;  \ RCC-
CICR_CSSLSEC    LSE Clock Security System Interrupt  clear
: RCC-CICR_HSI48RDYC   %1 6 lshift RCC-CICR bis! ;  \ RCC-
CICR_HSI48RDYC    HSI48 ready Interrupt  clear
: RCC-CICR_MSIRDYC   %1 5 lshift RCC-CICR bis! ;  \ RCC-
CICR_MSIRDYC    MSI ready Interrupt clear
: RCC-CICR_PLLRDYC   %1 4 lshift RCC-CICR bis! ;  \ RCC-
CICR_PLLRDYC    PLL ready Interrupt clear
: RCC-CICR_HSERDYC   %1 3 lshift RCC-CICR bis! ;  \ RCC-
CICR_HSERDYC    HSE ready Interrupt clear
: RCC-CICR_HSI16RDYC   %1 2 lshift RCC-CICR bis! ;  \ RCC-
CICR_HSI16RDYC    HSI16 ready Interrupt  clear
: RCC-CICR_LSERDYC   %1 1 lshift RCC-CICR bis! ;  \ RCC-
CICR_LSERDYC    LSE ready Interrupt clear
: RCC-CICR_LSIRDYC   %1 0 lshift RCC-CICR bis! ;  \ RCC-
CICR_LSIRDYC    LSI ready Interrupt clear

\ RCC-IOPRSTR (read-write)
: RCC-IOPRSTR_IOPHRST   %1 7 lshift RCC-IOPRSTR bis! ;  \
RCC-IOPRSTR_IOPHRST    I/O port H reset
: RCC-IOPRSTR_IOPDRST   %1 3 lshift RCC-IOPRSTR bis! ;  \
RCC-IOPRSTR_IOPDRST    I/O port D reset
: RCC-IOPRSTR_IOPCRST   %1 2 lshift RCC-IOPRSTR bis! ;  \
RCC-IOPRSTR_IOPCRST    I/O port A reset
: RCC-IOPRSTR_IOPBRST   %1 1 lshift RCC-IOPRSTR bis! ;  \
RCC-IOPRSTR_IOPBRST    I/O port B reset
: RCC-IOPRSTR_IOPARST   %1 0 lshift RCC-IOPRSTR bis! ;  \
RCC-IOPRSTR_IOPARST    I/O port A reset
```

```
\ RCC-AHBRSTR (read-write)
: RCC-AHBRSTR_CRYPRST   %1 24 lshift RCC-AHBRSTR bis! ;  \
RCC-AHBRSTR_CRYPRST    Crypto module reset
: RCC-AHBRSTR_RNGRST   %1 20 lshift RCC-AHBRSTR bis! ;  \
RCC-AHBRSTR_RNGRST    Random Number Generator module  reset
: RCC-AHBRSTR_TOUCHRST   %1 16 lshift RCC-AHBRSTR bis! ;  \
RCC-AHBRSTR_TOUCHRST    Touch Sensing reset
: RCC-AHBRSTR_CRCRST   %1 12 lshift RCC-AHBRSTR bis! ;  \
RCC-AHBRSTR_CRCRST    Test integration module  reset
: RCC-AHBRSTR_MIFRST   %1 8 lshift RCC-AHBRSTR bis! ;  \ RCC-
AHBRSTR_MIFRST    Memory interface reset
: RCC-AHBRSTR_DMARST   %1 0 lshift RCC-AHBRSTR bis! ;  \ RCC-
AHBRSTR_DMARST    DMA reset

\ RCC-APB2RSTR (read-write)
: RCC-APB2RSTR_DBGRST   %1 22 lshift RCC-APB2RSTR bis! ;  \
RCC-APB2RSTR_DBGRST    DBG reset
: RCC-APB2RSTR_USART1RST   %1 14 lshift RCC-APB2RSTR bis! ;
\ RCC-APB2RSTR_USART1RST    USART1 reset
: RCC-APB2RSTR_SPI1RST   %1 12 lshift RCC-APB2RSTR bis! ;  \
RCC-APB2RSTR_SPI1RST    SPI 1 reset
: RCC-APB2RSTR_ADCRST   %1 9 lshift RCC-APB2RSTR bis! ;  \
RCC-APB2RSTR_ADCRST    ADC interface reset
: RCC-APB2RSTR_TM12RST   %1 5 lshift RCC-APB2RSTR bis! ;  \
RCC-APB2RSTR_TM12RST    TIM22 timer reset
: RCC-APB2RSTR_TIM21RST   %1 2 lshift RCC-APB2RSTR bis! ;  \
RCC-APB2RSTR_TIM21RST    TIM21 timer reset
: RCC-APB2RSTR_SYSCFGRST   %1 0 lshift RCC-APB2RSTR bis! ;  \
RCC-APB2RSTR_SYSCFGRST    System configuration controller
reset

\ RCC-APB1RSTR (read-write)
: RCC-APB1RSTR_LPTIM1RST   %1 31 lshift RCC-APB1RSTR bis! ;
\ RCC-APB1RSTR_LPTIM1RST    Low power timer reset
: RCC-APB1RSTR_DACRST   %1 29 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_DACRST    DAC interface reset
: RCC-APB1RSTR_PWRRST   %1 28 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_PWRRST    Power interface reset
: RCC-APB1RSTR_CRSRST   %1 27 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_CRSRST    Clock recovery system  reset
: RCC-APB1RSTR_USBRST   %1 23 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_USBRST    USB reset
: RCC-APB1RSTR_I2C2RST   %1 22 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_I2C2RST    I2C2 reset
```

```
: RCC-APB1RSTR_I2C1RST   %1 21 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_I2C1RST    I2C1 reset
: RCC-APB1RSTR_LPUART1RST   %1 18 lshift RCC-APB1RSTR bis! ;
\ RCC-APB1RSTR_LPUART1RST    LPUART1 reset
: RCC-APB1RSTR_LPUART12RST   %1 17 lshift RCC-APB1RSTR bis! ;
\ RCC-APB1RSTR_LPUART12RST    UART2 reset
: RCC-APB1RSTR_SPI2RST   %1 14 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_SPI2RST    SPI2 reset
: RCC-APB1RSTR_WWDRST   %1 11 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_WWDRST    Window watchdog reset
: RCC-APB1RSTR_LCDRST   %1 9 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_LCDRST    LCD reset
: RCC-APB1RSTR_TIM6RST   %1 4 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_TIM6RST    Timer 6 reset
: RCC-APB1RSTR_TIM2RST   %1 0 lshift RCC-APB1RSTR bis! ;  \
RCC-APB1RSTR_TIM2RST    Timer2 reset

\ RCC-IOPENR (read-write)
: RCC-IOPENR_IOPHEN   %1 7 lshift RCC-IOPENR bis! ;  \ RCC-
IOPENR_IOPHEN    I/O port H clock enable  bit
: RCC-IOPENR_IOPDEN   %1 3 lshift RCC-IOPENR bis! ;  \ RCC-
IOPENR_IOPDEN    I/O port D clock enable  bit
: RCC-IOPENR_IOPCEN   %1 2 lshift RCC-IOPENR bis! ;  \ RCC-
IOPENR_IOPCEN    IO port A clock enable bit
: RCC-IOPENR_IOPBEN   %1 1 lshift RCC-IOPENR bis! ;  \ RCC-
IOPENR_IOPBEN    IO port B clock enable bit
: RCC-IOPENR_IOPAEN   %1 0 lshift RCC-IOPENR bis! ;  \ RCC-
IOPENR_IOPAEN    IO port A clock enable bit

\ RCC-AHBENR (read-write)
: RCC-AHBENR_CRYPEN   %1 24 lshift RCC-AHBENR bis! ;  \ RCC-
AHBENR_CRYPEN    Crypto clock enable bit
: RCC-AHBENR_RNGEN   %1 20 lshift RCC-AHBENR bis! ;  \ RCC-
AHBENR_RNGEN    Random Number Generator clock enable  bit
: RCC-AHBENR_TOUCHEN   %1 16 lshift RCC-AHBENR bis! ;  \ RCC-
AHBENR_TOUCHEN    Touch Sensing clock enable  bit
: RCC-AHBENR_CRCEN   %1 12 lshift RCC-AHBENR bis! ;  \ RCC-
AHBENR_CRCEN    CRC clock enable bit
: RCC-AHBENR_MIFEN   %1 8 lshift RCC-AHBENR bis! ;  \ RCC-
AHBENR_MIFEN    NVM interface clock enable  bit
: RCC-AHBENR_DMAEN   %1 0 lshift RCC-AHBENR bis! ;  \ RCC-
AHBENR_DMAEN    DMA clock enable bit

\ RCC-APB2ENR (read-write)
: RCC-APB2ENR_DBGEN   %1 22 lshift RCC-APB2ENR bis! ;  \ RCC-
```

```
APB2ENR_DBGEN    DBG clock enable bit
: RCC-APB2ENR_USART1EN   %1 14 lshift RCC-APB2ENR bis! ;  \
RCC-APB2ENR_USART1EN    USART1 clock enable bit
: RCC-APB2ENR_SPI1EN   %1 12 lshift RCC-APB2ENR bis! ;  \
RCC-APB2ENR_SPI1EN    SPI1 clock enable bit
: RCC-APB2ENR_ADCEN   %1 9 lshift RCC-APB2ENR bis! ;  \ RCC-
APB2ENR_ADCEN    ADC clock enable bit
: RCC-APB2ENR_MIFIEN   %1 7 lshift RCC-APB2ENR bis! ;  \ RCC-
APB2ENR_MIFIEN    MiFaRe Firewall clock enable  bit
: RCC-APB2ENR_TIM22EN   %1 5 lshift RCC-APB2ENR bis! ;  \
RCC-APB2ENR_TIM22EN    TIM22 timer clock enable  bit
: RCC-APB2ENR_TIM21EN   %1 2 lshift RCC-APB2ENR bis! ;  \
RCC-APB2ENR_TIM21EN    TIM21 timer clock enable  bit
: RCC-APB2ENR_SYSCFGEN   %1 0 lshift RCC-APB2ENR bis! ;  \
RCC-APB2ENR_SYSCFGEN    System configuration controller clock
enable bit

\ RCC-APB1ENR (read-write)
: RCC-APB1ENR_LPTIM1EN   %1 31 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_LPTIM1EN    Low power timer clock enable  bit
: RCC-APB1ENR_DACEN   %1 29 lshift RCC-APB1ENR bis! ;  \ RCC-
APB1ENR_DACEN    DAC interface clock enable  bit
: RCC-APB1ENR_PWREN   %1 28 lshift RCC-APB1ENR bis! ;  \ RCC-
APB1ENR_PWREN    Power interface clock enable  bit
: RCC-APB1ENR_CRSEN   %1 27 lshift RCC-APB1ENR bis! ;  \ RCC-
APB1ENR_CRSEN    Clock recovery system clock enable  bit
: RCC-APB1ENR_USBEN   %1 23 lshift RCC-APB1ENR bis! ;  \ RCC-
APB1ENR_USBEN    USB clock enable bit
: RCC-APB1ENR_I2C2EN   %1 22 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_I2C2EN    I2C2 clock enable bit
: RCC-APB1ENR_I2C1EN   %1 21 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_I2C1EN    I2C1 clock enable bit
: RCC-APB1ENR_LPUART1EN   %1 18 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_LPUART1EN    LPUART1 clock enable bit
: RCC-APB1ENR_USART2EN   %1 17 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_USART2EN    UART2 clock enable bit
: RCC-APB1ENR_SPI2EN   %1 14 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_SPI2EN    SPI2 clock enable bit
: RCC-APB1ENR_WWDGEN   %1 11 lshift RCC-APB1ENR bis! ;  \
RCC-APB1ENR_WWDGEN    Window watchdog clock enable  bit
: RCC-APB1ENR_LCDEN   %1 9 lshift RCC-APB1ENR bis! ;  \ RCC-
APB1ENR_LCDEN    LCD clock enable bit
: RCC-APB1ENR_TIM6EN   %1 4 lshift RCC-APB1ENR bis! ;  \ RCC-
APB1ENR_TIM6EN    Timer 6 clock enable bit
: RCC-APB1ENR_TIM2EN   %1 0 lshift RCC-APB1ENR bis! ;  \ RCC-
```

```
APB1ENR_TIM2EN    Timer2 clock enable bit

\ RCC-IOPSMEN (read-write)
: RCC-IOPSMEN_IOPHSMEN   %1 7 lshift RCC-IOPSMEN bis! ;  \
RCC-IOPSMEN_IOPHSMEN    IOPHSMEN
: RCC-IOPSMEN_IOPDSMEN   %1 3 lshift RCC-IOPSMEN bis! ;  \
RCC-IOPSMEN_IOPDSMEN    IOPDSMEN
: RCC-IOPSMEN_IOPCSMEN   %1 2 lshift RCC-IOPSMEN bis! ;  \
RCC-IOPSMEN_IOPCSMEN    IOPCSMEN
: RCC-IOPSMEN_IOPBSMEN   %1 1 lshift RCC-IOPSMEN bis! ;  \
RCC-IOPSMEN_IOPBSMEN    IOPBSMEN
: RCC-IOPSMEN_IOPASMEN   %1 0 lshift RCC-IOPSMEN bis! ;  \
RCC-IOPSMEN_IOPASMEN    IOPASMEN

\ RCC-AHBSMENR (read-write)
: RCC-AHBSMENR_CRYPSMEN   %1 24 lshift RCC-AHBSMENR bis! ;  \
RCC-AHBSMENR_CRYPSMEN    Crypto clock enable during sleep
mode  bit
: RCC-AHBSMENR_RNGSMEN   %1 20 lshift RCC-AHBSMENR bis! ;  \
RCC-AHBSMENR_RNGSMEN    Random Number Generator clock enable
during sleep mode bit
: RCC-AHBSMENR_TOUCHSMEN   %1 16 lshift RCC-AHBSMENR bis! ;
\ RCC-AHBSMENR_TOUCHSMEN    Touch Sensing clock enable during
sleep  mode bit
: RCC-AHBSMENR_CRCSMEN   %1 12 lshift RCC-AHBSMENR bis! ;  \
RCC-AHBSMENR_CRCSMEN    CRC clock enable during sleep mode
bit
: RCC-AHBSMENR_SRAMSMEN   %1 9 lshift RCC-AHBSMENR bis! ;  \
RCC-AHBSMENR_SRAMSMEN    SRAM interface clock enable during
sleep  mode bit
: RCC-AHBSMENR_MIFSMEN   %1 8 lshift RCC-AHBSMENR bis! ;  \
RCC-AHBSMENR_MIFSMEN    NVM interface clock enable during
sleep  mode bit
: RCC-AHBSMENR_DMASMEN   %1 0 lshift RCC-AHBSMENR bis! ;  \
RCC-AHBSMENR_DMASMEN    DMA clock enable during sleep mode
bit

\ RCC-APB2SMENR (read-write)
: RCC-APB2SMENR_DBGSMEN   %1 22 lshift RCC-APB2SMENR bis! ;
\ RCC-APB2SMENR_DBGSMEN    DBG clock enable during sleep mode
bit
: RCC-APB2SMENR_USART1SMEN   %1 14 lshift RCC-APB2SMENR bis!
;  \ RCC-APB2SMENR_USART1SMEN    USART1 clock enable during
sleep mode  bit
: RCC-APB2SMENR_SPI1SMEN   %1 12 lshift RCC-APB2SMENR bis! ;
```

```
\ RCC-APB2SMENR_SPI1SMEN    SPI1 clock enable during sleep
mode  bit
: RCC-APB2SMENR_ADCSMEN   %1 9 lshift RCC-APB2SMENR bis! ;  \
RCC-APB2SMENR_ADCSMEN    ADC clock enable during sleep mode
bit
: RCC-APB2SMENR_TIM22SMEN   %1 5 lshift RCC-APB2SMENR bis! ;
\ RCC-APB2SMENR_TIM22SMEN    TIM22 timer clock enable during
sleep  mode bit
: RCC-APB2SMENR_TIM21SMEN   %1 2 lshift RCC-APB2SMENR bis! ;
\ RCC-APB2SMENR_TIM21SMEN    TIM21 timer clock enable during
sleep  mode bit
: RCC-APB2SMENR_SYSCFGSMEN   %1 0 lshift RCC-APB2SMENR bis! ;
\ RCC-APB2SMENR_SYSCFGSMEN    System configuration controller
clock  enable during sleep mode bit

\ RCC-APB1SMENR (read-write)
: RCC-APB1SMENR_LPTIM1SMEN   %1 31 lshift RCC-APB1SMENR bis!
;  \ RCC-APB1SMENR_LPTIM1SMEN    Low power timer clock enable
during  sleep mode bit
: RCC-APB1SMENR_DACSMEN   %1 29 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_DACSMEN    DAC interface clock enable during
sleep  mode bit
: RCC-APB1SMENR_PWRSMEN   %1 28 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_PWRSMEN    Power interface clock enable
during  sleep mode bit
: RCC-APB1SMENR_CRSSMEN   %1 27 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_CRSSMEN    Clock recovery system clock enable
during sleep mode bit
: RCC-APB1SMENR_USBSMEN   %1 23 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_USBSMEN    USB clock enable during sleep mode
bit
: RCC-APB1SMENR_I2C2SMEN   %1 22 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_I2C2SMEN    I2C2 clock enable during sleep
mode  bit
: RCC-APB1SMENR_I2C1SMEN   %1 21 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_I2C1SMEN    I2C1 clock enable during sleep
mode  bit
: RCC-APB1SMENR_LPUART1SMEN   %1 18 lshift RCC-APB1SMENR bis!
;  \ RCC-APB1SMENR_LPUART1SMEN    LPUART1 clock enable during
sleep mode  bit
: RCC-APB1SMENR_USART2SMEN   %1 17 lshift RCC-APB1SMENR bis!
;  \ RCC-APB1SMENR_USART2SMEN    UART2 clock enable during
sleep mode  bit
: RCC-APB1SMENR_SPI2SMEN   %1 14 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_SPI2SMEN    SPI2 clock enable during sleep
```

```
mode  bit
: RCC-APB1SMENR_WWDGSMEN   %1 11 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_WWDGSMEN    Window watchdog clock enable
during  sleep mode bit
: RCC-APB1SMENR_LCDSMEN   %1 9 lshift RCC-APB1SMENR bis! ;  \
RCC-APB1SMENR_LCDSMEN    LCD clock enable during sleep mode
bit
: RCC-APB1SMENR_TIM6SMEN   %1 4 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_TIM6SMEN    Timer 6 clock enable during sleep
mode  bit
: RCC-APB1SMENR_TIM2SMEN   %1 0 lshift RCC-APB1SMENR bis! ;
\ RCC-APB1SMENR_TIM2SMEN    Timer2 clock enable during sleep
mode  bit

\ RCC-CCIPR (read-write)
: RCC-CCIPR_HSI48MSEL   %1 26 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_HSI48MSEL    48 MHz HSI48 clock source selection  bit
: RCC-CCIPR_LPTIM1SEL1   %1 19 lshift RCC-CCIPR bis! ;  \
RCC-CCIPR_LPTIM1SEL1    Low Power Timer clock source
selection  bits
: RCC-CCIPR_LPTIM1SEL0   %1 18 lshift RCC-CCIPR bis! ;  \
RCC-CCIPR_LPTIM1SEL0    LPTIM1SEL0
: RCC-CCIPR_I2C1SEL1   %1 13 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_I2C1SEL1    I2C1 clock source selection  bits
: RCC-CCIPR_I2C1SEL0   %1 12 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_I2C1SEL0    I2C1SEL0
: RCC-CCIPR_LPUART1SEL1   %1 11 lshift RCC-CCIPR bis! ;  \
RCC-CCIPR_LPUART1SEL1    LPUART1 clock source selection  bits
: RCC-CCIPR_LPUART1SEL0   %1 10 lshift RCC-CCIPR bis! ;  \
RCC-CCIPR_LPUART1SEL0    LPUART1SEL0
: RCC-CCIPR_USART2SEL1   %1 3 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_USART2SEL1    USART2 clock source selection  bits
: RCC-CCIPR_USART2SEL0   %1 2 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_USART2SEL0    USART2SEL0
: RCC-CCIPR_USART1SEL1   %1 1 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_USART1SEL1    USART1 clock source selection  bits
: RCC-CCIPR_USART1SEL0   %1 0 lshift RCC-CCIPR bis! ;  \ RCC-
CCIPR_USART1SEL0    USART1SEL0

\ RCC-CSR ()
: RCC-CSR_LPWRSTF   %1 31 lshift RCC-CSR bis! ;  \ RCC-
CSR_LPWRSTF    Low-power reset flag
: RCC-CSR_WWDGRSTF   %1 30 lshift RCC-CSR bis! ;  \ RCC-
CSR_WWDGRSTF    Window watchdog reset flag
: RCC-CSR_IWDGRSTF   %1 29 lshift RCC-CSR bis! ;  \ RCC-
```

```
CSR_IWDGRSTF    Independent watchdog reset  flag
: RCC-CSR_SFTRSTF   %1 28 lshift RCC-CSR bis! ;  \ RCC-
CSR_SFTRSTF    Software reset flag
: RCC-CSR_PORRSTF   %1 27 lshift RCC-CSR bis! ;  \ RCC-
CSR_PORRSTF    POR/PDR reset flag
: RCC-CSR_PINRSTF   %1 26 lshift RCC-CSR bis! ;  \ RCC-
CSR_PINRSTF    PIN reset flag
: RCC-CSR_OBLRSTF   %1 25 lshift RCC-CSR bis! ;  \ RCC-
CSR_OBLRSTF    OBLRSTF
: RCC-CSR_RMVF   %1 24 lshift RCC-CSR bis! ;  \ RCC-CSR_RMVF
Remove reset flag
: RCC-CSR_RTCRST   %1 19 lshift RCC-CSR bis! ;  \ RCC-
CSR_RTCRST    RTC software reset bit
: RCC-CSR_RTCEN   %1 18 lshift RCC-CSR bis! ;  \ RCC-
CSR_RTCEN    RTC clock enable bit
: RCC-CSR_RTCSEL   ( %XX -- ) 16 lshift RCC-CSR bis! ;  \
RCC-CSR_RTCSEL    RTC and LCD clock source selection  bits
: RCC-CSR_CSSLSED   %1 14 lshift RCC-CSR bis! ;  \ RCC-
CSR_CSSLSED    CSS on LSE failure detection  flag
: RCC-CSR_CSSLSEON   %1 13 lshift RCC-CSR bis! ;  \ RCC-
CSR_CSSLSEON    CSSLSEON
: RCC-CSR_LSEDRV   ( %XX -- ) 11 lshift RCC-CSR bis! ;  \
RCC-CSR_LSEDRV    LSEDRV
: RCC-CSR_LSEBYP   %1 10 lshift RCC-CSR bis! ;  \ RCC-
CSR_LSEBYP    External low-speed oscillator bypass  bit
: RCC-CSR_LSERDY   %1 9 lshift RCC-CSR bis! ;  \ RCC-
CSR_LSERDY    External low-speed oscillator ready  bit
: RCC-CSR_LSEON   %1 8 lshift RCC-CSR bis! ;  \ RCC-CSR_LSEON
External low-speed oscillator enable  bit
: RCC-CSR_LSIRDY   %1 1 lshift RCC-CSR bis! ;  \ RCC-
CSR_LSIRDY    Internal low-speed oscillator ready  bit
: RCC-CSR_LSION   %1 0 lshift RCC-CSR bis! ;  \ RCC-CSR_LSION
Internal low-speed oscillator  enable
```

Older STM32 Releases

Svd2forth-v1

Svd2forth-v1 uses a ARM CMSIS-SVD XML file, to produce a Mecrisp-Stellaris Memory Map and Register reference file for a STM32xxx MCU. Unlike svd2forth-v3-stm32, Svd2forth-v1 creates bare bones, minimal files with no extra pretty

Register printing Words. Svd2forth-v1 is for STM32 users who don’t want any extras.

STM32F0 with all Register Words loaded into Flash

Loading *all* the Register Words is *highly* unlikely in practice, this table just demonstrates it can be done.

MCU	Flash Size	Free after installing Mecrisp-Stellaris	Free after loading all registers memory map from SVD2Forth-v1 (15052 bytes)
STM32F0	64KB	44772 Bytes	29720

Svd2forth-v1 Example

memmap

STM32F0xx.svd.uf.svd.memmap.fs. Maps Registers to memory.

```
$40023000 constant CRC
CRC $0 +  constant  CRC_DR
CRC $4 +  constant  CRC_IDR
CRC $8 +  constant  CRC_CR
CRC $C +  constant  CRC_INIT

$48001400 constant GPIOF
GPIOF $0 +  constant  GPIOF_MODER
GPIOF $4 +  constant  GPIOF_OTYPER
GPIOF $8 +  constant  GPIOF_OSPEEDR
GPIOF $C +  constant  GPIOF_PUPDR
GPIOF $10 +  constant  GPIOF_IDR
GPIOF $14 +  constant  GPIOF_ODR
GPIOF $18 +  constant  GPIOF_BSRR
GPIOF $1C +  constant  GPIOF_LCKR
GPIOF $20 +  constant  GPIOF_AFRL
GPIOF $24 +  constant  GPIOF_AFRH
GPIOF $28 +  constant  GPIOF_BRR

...

$4001001C constant COMP
COMP $0 +  constant  COMP_CSR

...
```

register-reference

A programmers reference.

```
################################### CRC
###################################
CRC_DR (read-write)
$FFFFFFFF CONSTANT RESET_CRC_DR
%xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  0 lshift CRC_DR bis!
\ CRC_DR Bit 0    Width 32

CRC_IDR (read-write)
$00000000 CONSTANT RESET_CRC_IDR
%xxxxxxxx  0 lshift CRC_IDR bis!
\ CRC_IDR        Bit 0    Width 8

CRC_CR (read-write)
$00000000 CONSTANT RESET_CRC_CR
%1  0 lshift CRC_CR bis!
\ CRC_RESET      Bit 0    Width 1
%xx  5 lshift CRC_CR bis!
\ CRC_REV_IN     Bit 5    Width 2
%1  7 lshift CRC_CR bis!
\ CRC_REV_OUT    Bit 7    Width 1

CRC_INIT (read-write)
$FFFFFFFF CONSTANT RESET_CRC_INIT
%xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  0 lshift CRC_INIT bis!
\ CRC_INIT       Bit 0    Width 32


################################### GPIOF
```

```
###################################
GPIOF_MODER (read-write)
$00000000 CONSTANT RESET_GPIOF_MODER
%xx  30 lshift GPIOF_MODER bis!        \ GPIOF_MODER15
Bit 30     Width 2
%xx  28 lshift GPIOF_MODER bis!        \ GPIOF_MODER14
Bit 28     Width 2
%xx  26 lshift GPIOF_MODER bis!        \ GPIOF_MODER13
Bit 26     Width 2
%xx  24 lshift GPIOF_MODER bis!        \ GPIOF_MODER12
Bit 24     Width 2
%xx  22 lshift GPIOF_MODER bis!        \ GPIOF_MODER11
Bit 22     Width 2
%xx  20 lshift GPIOF_MODER bis!        \ GPIOF_MODER10
Bit 20     Width 2
%xx  18 lshift GPIOF_MODER bis!        \ GPIOF_MODER9
Bit 18     Width 2
%xx  16 lshift GPIOF_MODER bis!        \ GPIOF_MODER8
Bit 16     Width 2
%xx  14 lshift GPIOF_MODER bis!        \ GPIOF_MODER7
Bit 14     Width 2
%xx  12 lshift GPIOF_MODER bis!        \ GPIOF_MODER6
Bit 12     Width 2
%xx  10 lshift GPIOF_MODER bis!        \ GPIOF_MODER5
Bit 10     Width 2
%xx  8 lshift GPIOF_MODER bis!         \ GPIOF_MODER4
Bit 8      Width 2
%xx  6 lshift GPIOF_MODER bis!         \ GPIOF_MODER3
Bit 6      Width 2
%xx  4 lshift GPIOF_MODER bis!         \ GPIOF_MODER2
Bit 4      Width 2
%xx  2 lshift GPIOF_MODER bis!         \ GPIOF_MODER1
Bit 2      Width 2
%xx  0 lshift GPIOF_MODER bis!         \ GPIOF_MODER0
Bit 0      Width 2

...

################################### COMP
###################################
COMP_CSR ()
$00000000 CONSTANT RESET_COMP_CSR
%1  0 lshift COMP_CSR bis!           \ COMP_COMP1EN
Bit 0      Width 1
%1  1 lshift COMP_CSR bis!           \ COMP_COMP1_INP_DAC
```

```
Bit 1      Width 1
%xx  2 lshift COMP_CSR bis!          \ COMP_COMP1MODE
Bit 2      Width 2
%xxx  4 lshift COMP_CSR bis!         \ COMP_COMP1INSEL
Bit 4      Width 3
%xxx  8 lshift COMP_CSR bis!         \ COMP_COMP1OUTSEL
Bit 8      Width 3
%1  11 lshift COMP_CSR bis!          \ COMP_COMP1POL
Bit 11     Width 1
%xx  12 lshift COMP_CSR bis!         \ COMP_COMP1HYST
Bit 12     Width 2
%1  14 lshift COMP_CSR bis!          \ COMP_COMP1OUT
Bit 14     Width 1
%1  15 lshift COMP_CSR bis!          \ COMP_COMP1LOCK
Bit 15     Width 1
%1  16 lshift COMP_CSR bis!          \ COMP_COMP2EN
Bit 16     Width 1
%xx  18 lshift COMP_CSR bis!         \ COMP_COMP2MODE
Bit 18     Width 2
%xxx  20 lshift COMP_CSR bis!        \ COMP_COMP2INSEL
Bit 20     Width 3
%1  23 lshift COMP_CSR bis!          \ COMP_WNDWEN
Bit 23     Width 1
%xxx  24 lshift COMP_CSR bis!        \ COMP_COMP2OUTSEL
Bit 24     Width 3
%1  27 lshift COMP_CSR bis!          \ COMP_COMP2POL
Bit 27     Width 1
%xx  28 lshift COMP_CSR bis!         \ COMP_COMP2HYST
Bit 28     Width 2
%1  30 lshift COMP_CSR bis!          \ COMP_COMP2OUT
Bit 30     Width 1
%1  31 lshift COMP_CSR bis!          \ COMP_COMP2LOCK
Bit 31     Width 1

...
```

Usage Example

You want to set bit 15 of GPIOF as a OUTPUT, the STM32F0 Reference tells you that MODER15 is the GPIO BitField you need and that it must be set to %01. Searching for for “moder15” in the STM32F0xx.svd.uf.svd.register-reference.fs file soon gives you the following line with example code to SET this BitField with “bis” (Bit Set).

```
%xx  30 lshift GPIOF_MODER bis!        \ GPIOF_MODER15
Bit 30     Width 2
```

So your code to SET bit 15 of GPIOF to a OUTPUT is :-

```
%01 30 lshift GPIOF_MODER bis!     \ Set bit 15 of GPIOF to a
OUTPUT
```

The Mecrisp REPL

Terminal handlers

emit, emit?, key and key? are the UART handlers.

If you wish to use blocking IO, you can set hook-emit? and hook-key? statically to true.

Repl Maximum Character Length

You can change the accepted length in forth-core.s for “Maximaleeingabe” which is currently set to 200 characters.

Ledcomm Example from Matthias

https://sourceforge.net/p/mecrisp/discussion/general/thread/e29549a8d5/?limit=25#4afd/6b1e [matthias.koch]

```
key? ( -- ? ) is giving you a flag meaning if there is (at least) one character in the receive buffer.
key ( -- c ) is waiting until a character is received (it calls key? internally in a loop) and gives back that character.

The hooks are just vectors, accepting a pointer to a definition.

hook-key takes a definition which works as "key" and hook-key? takes a definition which works as "key?"

Look at this snipplet:

0 variable tx-puffer
0 variable rx-puffer
```

```
Communication is done from/to buffer variables driven by
timers/interrupts

Then wire the buffers in accordingly:

: ledcomm-key? ( -- ? ) rx-puffer @ 0<> ;
: ledcomm-key ( -- c ) begin ledcomm-key? until rx-puffer @ 0
rx-puffer ! ;
: ledcomm-emit? ( -- ? ) tx-puffer @ 0= ;
: ledcomm-emit ( c -- ) begin ledcomm-emit? until tx-puffer !
;

Switch terminal to new definitions:

['] ledcomm-key? hook-key? !
['] ledcomm-key hook-key !
['] ledcomm-emit? hook-emit? !
['] ledcomm-emit hook-emit !
```

> What’s New ? <

Serial Terminal: Software

Serial terminal software requires Serial Terminal Hardware

Organised with my preferred Forth Serial Terminals first.

GnuScreen

https://www.gnu.org/software/screen/

GnuScreen is my current serial terminal of choice as of September 2018.

e4thcom-0.6.1

> E4thcom, Copyright (C) 2013-2017 Manfred Mahlow and licensed under the GP. https://wiki.forth-ev.de/doku.php/en:projects:e4thcom#e4thcom-061

- Advantages
    - Strips comments from the source code
    - Conditional and unconditional uploading of source code via # require directives.I can’t stress how **important** this one feature is as it allows all the forth code dependencies in your project to be automatically uploaded to the target in the order you specify.
    - Files already uploaded with “#r” or “#require” will be ignored if they are uploaded again (via a Dictionary check) , preventing Word redefinition memory wastage.

  - If uploading is aborted with an error message, typing #edit [Enter] or #e [Enter] at the terminal will open the uploaded file with the cursor placed in the line that raised the error.
  - One can create a file in your editor with the #edit [Enter] or #e [Enter at the terminal. the dotfile ~/.e4thcom-0.6.1 needs to know about your editor.
  - Mecrisp-Stellaris plug-in for upload flow control and error handling.This means that without any flow control, uploading will be as fast as possible, no delays!
  - Editable command line buffer with history and tab selection (/index) No more re-typing commands
  - Predefined code path searching for libraries and mcu. A neat and organised file project hierarchy

- Disadvantages
  - Linux only

Operating system availability

E4thcom is supplied as a binary release for:-

1. Linux
2. Raspberry Pi

If you have FreeBSD like I do, then you can use it via a RaspberryPI or Linux Emulation (not tried).

Get the binary release for this URI: https://wiki.forth-ev.de/lib/exe/fetch.php/projects:e4thcom:e4thcom-0.6.1.tar.gz

Get the e4thcom-0.6.1 **`PDF User Manual`** here.

Using e4thcom

File Structure

Start E4thcom in your project directory, and make sure that any libraries are located in lib/ in that directory. E4thsom will search the project directory and the lib/ directory when you instruct it to require any files. See the E4thcom doc in the release for more information.

Set your *editor* in .e4thcom-0.6.1 which should be located in your home directory after copying it there from the release tarball.

Starting e4thcom on a RaspberryPi

Note

Be sure to use **your** serial port device when connecting to Mecrisp-Stellaris.

Note

For Mecrisp-Stellaris the option **-t mecrisp-st** must be used. The -t mecrisp option will only work as long as no resource files are used.

```
e4thcom -t mecrisp-st -d ttyUSB0 -b B115200
```

Logging

Logging e4thcom is easily done in Unix like so:-

```
e4thcom -t mecrisp-st -d ttyUSB0 -b B115200  |  tee
e4thcom.log
```

Picocom

- Advantages
    - A useful serial terminal for quick tests: https://github.com/npat-efault/picocom
- Disadvantages (note: Picocom is a excellent, stable and easy to use terminal)
    - Can be *painfully* slow to upload files as the line delay must be set to the *maximum* required delay, which also slows down short and fast Forth command uploads.
    - Uploads comments which must then be discarded by the Forth system
    - No way to automatically upload dependent files as E4thcom does using #require or #include.

Picocom, no handshaking, end of line delay of 200ms

```
picocom -b 115200 /dev/cuaU0 --imap lfcrlf,crcrlf --omap
delbs,crlf --send-cmd "ascii-xfr -s -l200"
```

Picocom, 460800 Baud hardware rts/cts handshaking, no end of line delay, no double spacing of received data

```
picocom -f h -b 460800 /dev/cuaU0 --omap delbs,crlf --send-
cmd "ascii-xfr -s"
```

Picocom Shell Script with xterm title

```
xterm -geometry 108x75+1+1 -fs 10 -title 'Picocom 460800,
RTSCTS, /dev/cuaU0 |  <ctrl> AQ to exit ' -e picocom -f h -b
```

```
460800 /dev/cuaU0 --omap delbs,crlf --send-cmd "ascii-xfr -s"
&
```

Picocom Command Line details

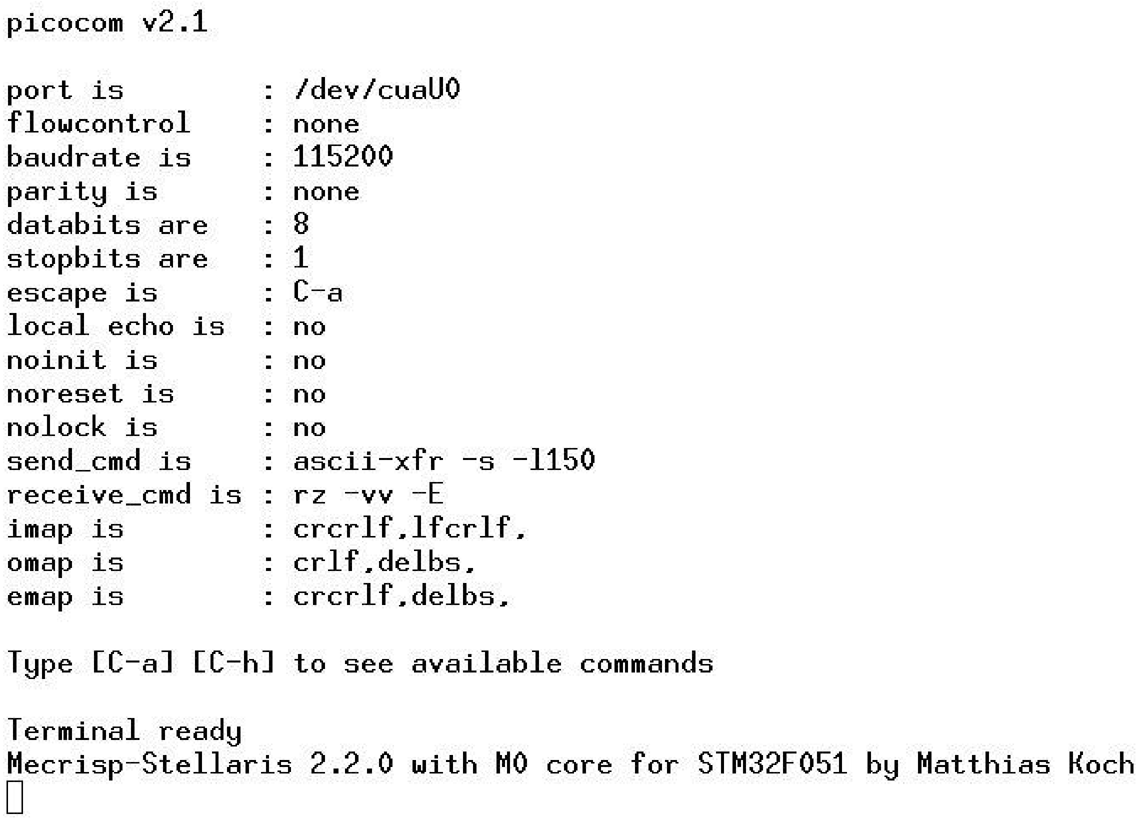
Command	Description
picocom	excutable
-b 115200	baud rate of 115200
-f h	hardware rts/cts handshaking
/dev/cuaU0	the PC serial device, yours may differ
–imap lfcrlf,crcrlf	input char mapping
–omap delbs,crlf	output char mapping
–send-cmd “ascii-xfr -s -l200	200mS line delay to give the compiler time when uploading files

Some useful Picocom Commands

Action	Description	Notes

Action	Description	Notes
quit	<ctrl> A Q	
upload a file	<ctrl> A S	this command has TAB completion

screenshot



```
picocom v2.1

port is        : /dev/cuaU0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : ascii-xfr -s -l150
receive_cmd is : rz -vv -E
imap is        : crcrlf,lfcrlf,
omap is        : crlf,delbs,
emap is        : crcrlf,delbs,

Type [C-a] [C-h] to see available commands

Terminal ready
Mecrisp-Stellaris 2.2.0 with M0 core for STM32F051 by Matthias Koch
```

Forth Explorer

Forth Exlorer was a quick hack especially for Mecrisp-Stellaris by K.Foltman and is similar to e4thcom, but lacks the vital Forth file #include facilities and others. Forth Explorer is now unsupported.

- Advantages
    - No end of line delay needed: FE waits for the ‘ok’ emitted by Mecrisp-Stellaris before sending the next line.Uploading Forth programs doesn’t get any faster than this.
    - Previous Command Recall: up arrow recalls previous commands from your session for reuse, a BIG timesaver, and FE now saves this between sessions so when you use FE next time, commands used in earlier sessions are still available with the up arrow.
    - Error checking: will stop on the most errors, alerting you to that “;” you forgot ;-) including :-
        - Not Found
        - Redefine
        - Unterminated string constant
        - Structures don’t match
    - Audible Error Alert: FE now plays configurable sound bites on *successful* file upload or *failure* :
        - file error audible alert
        - file completion audible alert
    - Comment stripping to speed things up.
    - Dumb Terminal Mode, switch between Smart and Dumb terminal on the fly !
- Disadvantages
    - No way to automatically upload dependent files as E4thcom does.

Note

FE Dependencies

- Python
- py-serial
- sox
- a couple of wav files to play for completed and error uploads, (set these in FE itself).

FE commands

Note: See the GNU Readline manual for all available commands

Description	Notes
python fe.py /dev/cuaU1	starts the FE terminal emulator in Smart mode
Ctrl + D	quit FE, (only in Smart mode)
!S <file> or Ctrl+ R	upload a file, defaults to last file
!T or Ctrl+ T	switch to a dumb TTY mode
Ctrl+ X	switch back to Smart Terminal mode

Download the **`Forth Explorer (FE)`** tarball with sound files.

Source Documentation

My embedded Forth code can be pretty hard to read, especially when it’s full of CMSIS-SVD named register configs and binary numbers so as I’ve been developing my Mecrisp-Stellaris IDE and learning Forth I have also been experimenting with different documentation strategies.

Now there are a few “Forth Style Guides” around but they just don’t work for me, so in true Forth tradition, I’m trying to create what works best in my case.

In-Source Text : Configuration Code

It’s horrible, I agree. This code is all initialisation code, barely a step up from C headers and about as interesting. I haven’t been able to make this kind of source any prettier yet. Does it actually need to be ?

Each line has a comment automatically pulled from the CMSIS-SCD ‘description field’, and often additional comments of mine but for deeper knowledge you’ll have to consult the STM32F0 Reference Manual. ALL the information you need to do that is in this code, however embedded programming requires a intimate knowledge of the MCU peripheral hardware.

```
\ Tim6 related Words
: nvic_iser_setena-tim6 ( -- ) 1 17 lshift nvic_iser bis! ;
\ nvic_iser_setena   tim6
: nvic_icer_clrena-tim6 ( -- ) 1 17 lshift nvic_icer bis! ;
\ nvic_icer_clrena  tim6
: nvic_icpr_clrpend-tim6 ( -- ) 1 17 lshift nvic_icpr bis! ;
\ nvic_icpr_clrpend  tim6
: tim6_cnt@ ( -- ) tim6_cnt h@ ;  \ tim6_cnt_cnt   16 bit
```

```
counter value
: tim6_cr1_opm ( -- ) %1 3 lshift tim6_cr1 hbis! ;      \
tim6_cr1_opm one-pulse mode
: tim6_cr1_cen ( -- ) %1 0 lshift tim6_cr1 hbis! ;      \
tim6_cr1_cen    counter enable
: tim6_cr1_cdis ( -- ) %1 0 lshift tim6_cr1 hbic! ;     \
tim6_cr1_cen    counter disable
: tim6_dier_uie ( -- ) %1 0 lshift tim6_dier hbis! ;    \
tim6_dier_uie        update interrupt enable
: tim6_sr_uif ( -- ) %1 0 lshift tim6_sr hbic! ;        \
tim6_sr_uif  clear tim6 int pending flag
: tim6_egr_ug ( -- ) %1 0 lshift tim6_egr hbis! ;       \
tim6_egr_ug  update generation

: tim6.prescale ( low-auto-reload-value -- ) \ Don't cause
interrupt. PSC does not get 'active' until the second update
event
  tim6_arr h!              \ low auto-reload value
  tim6_egr_ug              \ update generation: (causes a
interrupt when run) but loads ARR into shadow register!
  tim6_sr_uif              \ clear tim6 int pending flag
after running tim6_egr_ug
  tim6_dier_uie            \ update interrupt enable
;

: tim6.int.handler ( -- )
  nvic_icpr_clrpend-tim6                 \ clear pending iser
for tim6
  tim6_sr_uif                            \ clear tim6 int
pending flag
  tim6_cr1_cdis                          \ counter disable
  nvic_icer_clrena-tim6                  \ disable tim6
interrupt or dropping power to the LMT01 will re-trigger it
  green-off                              \ END COUNT WINDOW
  lmt01-x.power.off
  lmt01-x.count @ total.pulses.counted ! \
total.pulses.counted now contains valid lmt01 count
  1 ready.flag !                         \ it's safe to read
total.pulses.counted now
  nvic_iser_setena-tim6                  \ re-enable tim6
iser
  nvic_iser_setena-exti0_1               \ re enable exti0_1
iser ( won't work without this )
;
```

In-Source Text : Program Code

I’m pretty happy with this method of documentation and if the names of Words called are very descriptive, the source is almost self documenting which has always been a Forth feature.

```
: read.sensor ( sensor # -- error | stable data )      \
Select and read sensor, test for error
   0 lmt01-x.count !
   0 total.pulses.counted !
   0 ready.flag !
      case                                     \ power up
selected sensor
        1 OF lmt01-1.power.on ENDOF
        2 OF lmt01-2.power.on ENDOF
        3 OF lmt01-3.power.on ENDOF
        ." No such sensor !! "                 \ sensor not
implemented fall thru warning
      endcase
   green-on                                    \ START COUNT
WINDOW
   tim6_cr1_cen                                \ enable
timer6 (end of timer kills the power to the lmt01)
      begin                                    \ loop until
total.pulses.counted is stable
        pause                                  \ 'pause'
hands control back to the cooperative multitasker
        ready.flag @ 1 =                       \ has
ready.flag been set by tim6.int.handler ?
      until                                    \
total.pulses.counted now has valid count and can be read
      lmt01-fault? if ." lmt01-fault ! "
      then
 ;
```

Flowcharting

I’m exploring using Flowcharting for Forth PROGRAM documentation. Something that explains what all the sourcecode does in one big picture. It takes ages, especially after the fact and when trying to work out what the hell I wrote several months ago, so I’m trying to make flowcharting the INITIAL part of my Forth design process.

But here is the CATCH ..

I love just jumping straight into Forth coding, I mean that’s the beauty of the Forth paradigm, “try it before you buy it”. By this I mean with Forth I can just change some registers in near real time and observe the effect, be it turning a stepper motor or checking the byte order of a CRC generating peripheral. Using this method my code just seems to build itself.

The C methodology is studying the databook with infinite care, meticulously craft all my code, which then bombs because I misread the databook or it was ambiguous. So out with the debugger or scrutinise my code, then rinse and repeat.

wierd fact

When I used to program only in Machine Code (in the early 80’s), I flowcharted my programs first and the two technologies seemed to smoothly meld together. When I finally obtained a decent Motorola 6800 Assembler it was exactly the same, but when I moved up to C programming I never flowcharted again, I just couldn’t do it, some mental block stopped me. Perhaps C and flowcharts just don’t work together, at least in *my head* ?

So combining Flowcharting with Forth source code development in a easy and fun way is my current challenge.

Graphviz

I flowchart with Graphviz Dot and it’s built into my Forth development system. When I hit Make after editing the flowchart source, it’s compiled into a jpeg and the picture of the flowchart displayed for my review.

There are a few Diagrammers that run under Unix that use Java or C with some widget set that are mouse driven but I find them all too tedious and even more work with complex flowcharts than Graphviz.

I’m finding the Graphviz flowchart source a bit tedious also but I like the results. What do you think ?

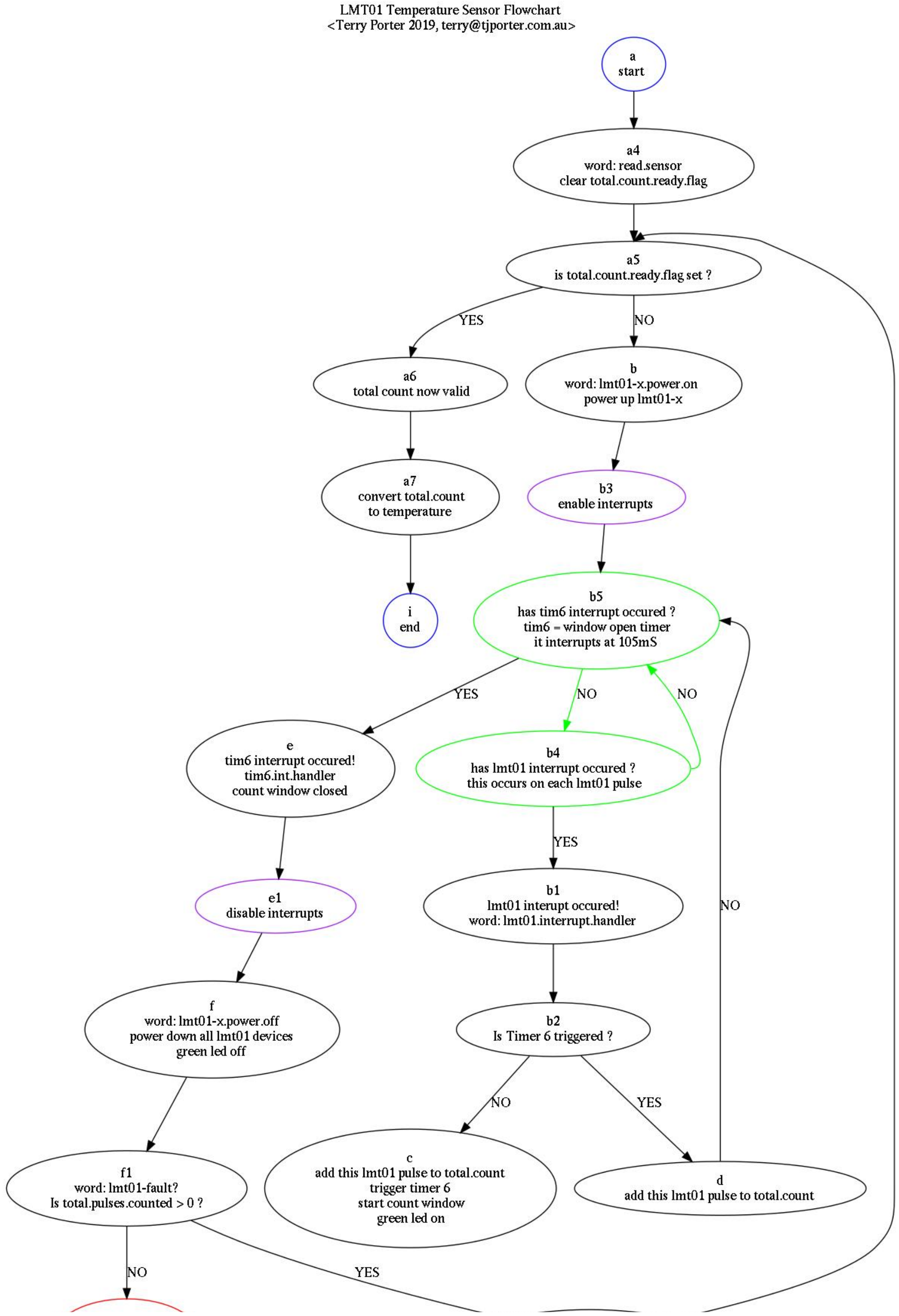
Flowchart Source Code

```
digraph {
// title
    labelloc="t";
    label="LMT01 Temperature Sensor Flowchart\n <Terry Porter 2019, terry@tjporter.com.au>";

A [color=blue,label="a\nstart "]
A4[label="a4\nword: read.sensor\n clear total.count.ready.flag "]
A5[label="a5\n is total.count.ready.flag set ? "]
A6[label="a6\n total count now valid"]
A7[label="a7\n convert total.count\nto temperature"]
B [label="b\nword: lmt01-x.power.on \npower up lmt01-x"]
B1[label="b1\nlmt01 interupt occured!\nword: lmt01.interrupt.handler"]
B2[label="b2\nIs Timer 6 triggered ?"]
B3[color=purple,label="b3\nenable interrupts"]
B4[color=green,label="b4\nhas lmt01 interrupt occured ?\n this occurs on each lmt01 pulse"]
B5[color=green,label="b5\nhas tim6 interrupt occured ?\n tim6 = window open timer\n it interrupts at 105mS "]
C [label="c\nadd this lmt01 pulse to total.count\ntrigger timer 6\n start count window\n green led on"]
D [label="d\nadd this lmt01 pulse to total.count"]
E [label="e\ntim6 interrupt occured!\ntim6.int.handler\ncount window closed"]
E1[color=purple,label ="e1\ndisable interrupts"]
F [label="f\nword: lmt01-x.power.off\npower down all lmt01
devices\ngreen led off"]
F1[label="f1\nword: lmt01-fault?\nIs total.pulses.counted > 0
?"]
G [color=red,label="g\nsensor faulty or\nnot present"]
H [label="h\nset total.count.ready.flag"]
I [color=blue,label="i\nend"]
J [color=red,label="j\nfault"]
A    -> A4
A4   -> A5
A5   -> B  [label="NO"]
A5   -> A6:n [label="YES"]
A6   -> A7
A7   -> I
B    -> B3
B1   -> B2
B2   -> C  [label="NO"]
B2   -> D  [label="YES"]
B3   -> B5
B4   -> B1 [label="YES"]
B4:e -> B5 [color=green,label="NO"]
B5   -> B4 [color=green,label="NO"]
B5   -> E  [label="YES"]
D    -> B5:e [label="NO"]
E    -> E1
E1   -> F
F    -> F1
F1   -> G   [label="NO"]
F1   -> H   [label="YES"]
H    -> A5:n
G    -> J
}
```

Flowchart Picture

This picture is compiled automatically by “dot” and 99% of the time it’s a great layout. On occasion I have to force a line to a particular node ‘edge’ but that’s rare and I try not to do it. Having said that, there are three such instances here, but I wanted a pleasing graph for this page :)

I don’t position any of the nodes, ever. I leave that entirely up to Graphviz. The whole idea is for super fast flowchart generation and minimal coding to get it.


LMT01 Temperature Sensor Flowchart
<Terry Porter 2019, terry@tjporter.com.au>
a
start
a4
word: read.sensor
clear total.count.ready.flag
a5
is total.count.ready.flag set ?
YES
NO
a6
total count now valid
b
word: lmt01-x.power.on
power up lmt01-x
a7
convert total.count
to temperature
b3
enable interrupts
i
end
b5
has tim6 interrupt occured ?
tim6 = window open timer
it interrupts at 105mS
YES
NO
NO
e
tim6 interrupt occured!
tim6.int.handler
count window closed
b4
has lmt01 interrupt occured ?
this occurs on each lmt01 pulse
YES
e1
disable interrupts
b1
lmt01 interupt occured!
word: lmt01.interrupt.handler
NO
f
word: lmt01-x.power.off
power down all lmt01 devices
green led off
b2
Is Timer 6 triggered ?
NO
YES
f1
word: lmt01-fault?
Is total.pulses.counted > 0 ?
c
add this lmt01 pulse to total.count
trigger timer 6
start count window
green led on
d
add this lmt01 pulse to total.count
NO
YES

g
sensor faulty or
not present
h
set total.count.ready.flag
j
fault

Special Chip?

Where To start ?

You have a favorite MCU that you'd like to run Mecrisp on, but it's not in the supported hardware list, where do you start ?

What Language is Mecrisp Written in ?

Every MCU listed in the supported hardware list is written by [matthias.koch] entirely in Assembly Language.

Methods for adding a new MCU break down into two groups

1) Similar To one in the Supported Hardware List ?

Are you sure that your MCU variant doesn't work already ?

Just because it’s not listed doesn’t mean it won’t work, just that no one has reported it working already. Matthias has a limited hardware selection so unless you send him a board with your mcu ready to test, then you will have to do the testing.

Testing your board is usually as easy as flashing the binary image from the family member that is on the supported hardware list into your MCU and seeing if the:

```
Mecrisp-Stellaris X.X.X for <chiptype> by Matthias Koch
```

greeting appears on your serial terminal.

No Greeting Message

Look into the USART code (written in Assembly Language), usually named “terminal.s” in your target directory. It’s possible the USART in your variant is slightly different, uses different pins, is used for some other purpose, has a different Baudrate setting etc. See “Building Mecrisp-Stellaris” below

Not a Assembly Code kind of person ? see “Sending Matthias Koch a Board” below.

Garbled Greeting Message

Almost working! try a different Baud rate on your serial terminal.

2) Entirely New Architecture ?

Mecrisp will need to be ported to your MCU architecture. If you’re not into Assembly Language, see “Sending Matthias Koch a Board” below, but this is really a last resort as

Matthias favourite way to get to new ports up and running is to offer his assistance to someone who wishes to do so.

Building Mecrisp-Stellaris

Building is usually done with the Makefile in the mecrisp-stellaris-source target subfolder. If you are using Debian, you should get everything necessary to do so by installing “apt-get install build-essential binutils-arm-none-eabi” For a complete package rebuild, run the “release” script, which has more dependencies, though, as the assembled cores get emulated in order to precompile Forth code.

Every MCU in the supported hardware list has its own sub directory and from there, the Mecrisp-Stellaris binary image is compiled.

Your PC must have the required applications installed, such as Gcc, Bash etc, and some minor script changes may be needed depending on your OS.

For instance I use FreeBSD and don’t have the utility “md5sum” as required by Mecrisp-Stellaris, but I do have “md5” so I’ve added a symlink on my pc for md5sum –> md5. Or I can just change the name to “md5” in the distribution.

Compiling all the supported chip types in the one operation is done by executing the “release” script in the distribution root directory.

Individual chips may be compiled in their own directory located in “mecrisp-stellaris-source” by executing “make”.

Sending Matthias Koch a Board

This is really a last resort, but it doesn’t hurt to ask. Like the rest of us he is very busy and may not even have the time to reply for weeks.

If you want to give [matthias.koch] a development board with your MCU to test/port, email asking if he is interested. If he is then include some postcards from your area with the board (which won’t be returned to you). There is no timeframe, but if it has heavy analogs, marvellous ADCs/DACs in combination with a lot of RAM, I know Matthias won’t be able to resist looking at it for long :)

My preferred MCU is the STM32F051 and so I sent Matthias a STM32F0 Discovery Board back in May 2014. It was soon running Mecrisp-Stellaris, but at that time, mine was the first STM32F0 MCU to be supported (V2.00).

Note

There is no guarantee that Matthias will accept a board, especially if he doesn’t like the architecture such as Z80, or STM8. Assembly Wizards are like artists, they won’t and can’t use bad paint …

Hacking Mecrisp-Stellaris

Forth Users are very diverse range of individuals and love to customise their Forth to best suit their own needs … which are usually quite different from other Forth Users.

Forth is *all about customisation* in my opinion.

I recommend the following methods when you begin to hack Mecrisp-Stellaris, because once you start …

Use a SCM

Version your MCU directory (or the entire Mecrisp-Stellaris distribution. I use Fossil SCM, but a wide range of SCM’s are available, i.e. Git, bzr, mecurial etc.

A SCM will avoid the inevitable 3000 uniquely modified copies of Mecrisp-Stellaris and make life much, much easier.

Scripts

Create a bunch of shell scripts to do the most repetitive tasks.

My Mecrisp-Stellaris Hacks

These are my favorite Mecrisp-Stellaris Kernel hacks.

Hardware handshaking

Because Forth has a “COC” (Compiler on Chip) the compiler can “choke up” from being fed Source Code faster than it can digest so I add RTS Hardware Handshaking which ensures a fast orderly flow.

Faster Serial

The default serial speed of 115200 Baud isn’t nearly fast enough for me, so I use 460,800 Baud. If Forth was not as fast, or faster that using C, Gcc, and OpenOCD with GDB, for Embedded Development I wouldn’t use it.

Speed is even more important for Forth because Forth Users have immediate nearly realtime access to the hardware. If I want to light a led on the target hardware, I can do it *right now at the terminal*, I don’t have to compile a test program and download it like a C Programming User.

COC hardware Error Detection

Traditionally Forth users watched the Source Code slowly upload via their 9600 Baud terminal and read back any errors. This can be very time consuming, error prone and frustrating at 9600 Baud, but absolutely impossible at 460800 Baud!

However since the Mecrisp-Stellaris COC knows when it sees a Source Code Error I use that with RTS Handshaking to halt the Source Code upload right at the *ERROR LINE*.

I also add code that makes a LED turn on when a Source Code Error is detected by the COC.

Gnu Screen

I believe that Gnu Screen is the best serial terminal for Forth because it can send commands to Mecrisp Stellaris plus upload source files, all under control of a script, editor and so on.

But GnuScreen has one annoying issue, namely it ‘Stair Steps’ the data from Mecrisp Stellaris and ignores any configuration to stop it. The issue is that Mecrisp-Stellaris sends only the traditional Unix *newline* (NL) character at the end of line, but GnuScreen *must have* both a *newline* (NL) and a *carriage return* (CR) just like DOS.

So I add a CR to the NL in Mecrisp-Stellaris which solves this issue.

Word Hacks

I prefer a simpler stack print word “.s”

Default Stack Print Word

```
.s Stack: [4 ] 42 2 3 4  TOS: 5  *>
```

My Stack Print Word

```
.s Stack: [4 ] 42 2 3 4 5
```

Stair-Stepping

Does your terminal output look like this ?

```
ok.
     ok.
          ok.
               ok.
                  reset Mecrisp-Stellaris RA 2.4.6 with M0
core for STM32F051 by Matthias Koch

ok.

ok.

ok.

ok.

ok.

ok.

ok.
```

If it does, then you’re sufferring from ‘stair stepping”.

This happens when a Serial Terminal refuses to return the “Carriage” (Cursor) after a “New Line” *NL* (also known as a Line Feed *LF*) is received. In the ‘picture’ above (taken from my system) all that’s happening after I hit the Carriage Return key, is a New Line.

A special plea to the author, Matthias please add a cr to your release by default ?

The Solution ?

Fortunately because Mecrisp-Stellaris is Open Source, there are numerous solutions to this issue. Look at it as a exercise for the new Forth user ?

Use a Terminal that is configurable for it

Picocom is given as a example, but other serial terminals have options regardingf CR/LF also. Some like E4thcom (linux only) deal with this by design and is FAR BETTER for a quick fix, highly recommended if you run Linux.

Picocom

Picocom has a command line syntax that solves the problem. It runs under all Unixes and Mac etc.

```
picocom -b 115200 /dev/cuaU0 --imap lfcrlf,crcrlf --omap
delbs,crlf --send-cmd "ascii-xfr -s -l200"
```

E4thcom

E4thcom fixes this issue for Mecrisp-Stellaris but only under Linux.

Add this code to your program in the INIT Word

This “hook” tells Mecrisp Stellaris to add a CR to the LF’s.

```
: emit-crlf ( c -- ) dup 10 = if 13 serial-emit then serial-
emit ;
' emit-crlf hook-emit !



Mecrisp-Stellaris RA 2.4.6 with M0 core for STM32F051 by
Matthias Koch
```

```
ok.

ok.

ok.

: emit-crlf ( c -- ) dup 10 = if 13 serial-emit then serial-
emit ;  ok.

ok.
   ok.
      ' emit-crlf hook-emit !  ok.
 ok.
 ok.
 ok.
```

Modify your Mecrisp-Stellaris Kernel

Change a few commands in the Mecrisp-Stellaris source file “datastackandmacros.s”

Change occurances of:

```
"\Meldung\n"

to

"\Meldung\r\n"
```

And rebuild the distribution by running “release” in the root directory.

Stacks

What is the default Stack size ?

Stack	Default Size
Data	64 Bytes
Return	64 Bytes

Can Stack Size Be Changed?

Yes, see kernel configurations

Stack Sensitive Prompt

Data left on the stack is a common bug, it’s inevitable and was a issue for me when I was learning Forth. Yes … it still happens now, but it’s so infrequent it really catches me out when it occurs.

I began to add “.s” to the end of every new Word just to be sure the Stack was clean, and that worked fine but was a bit messy.

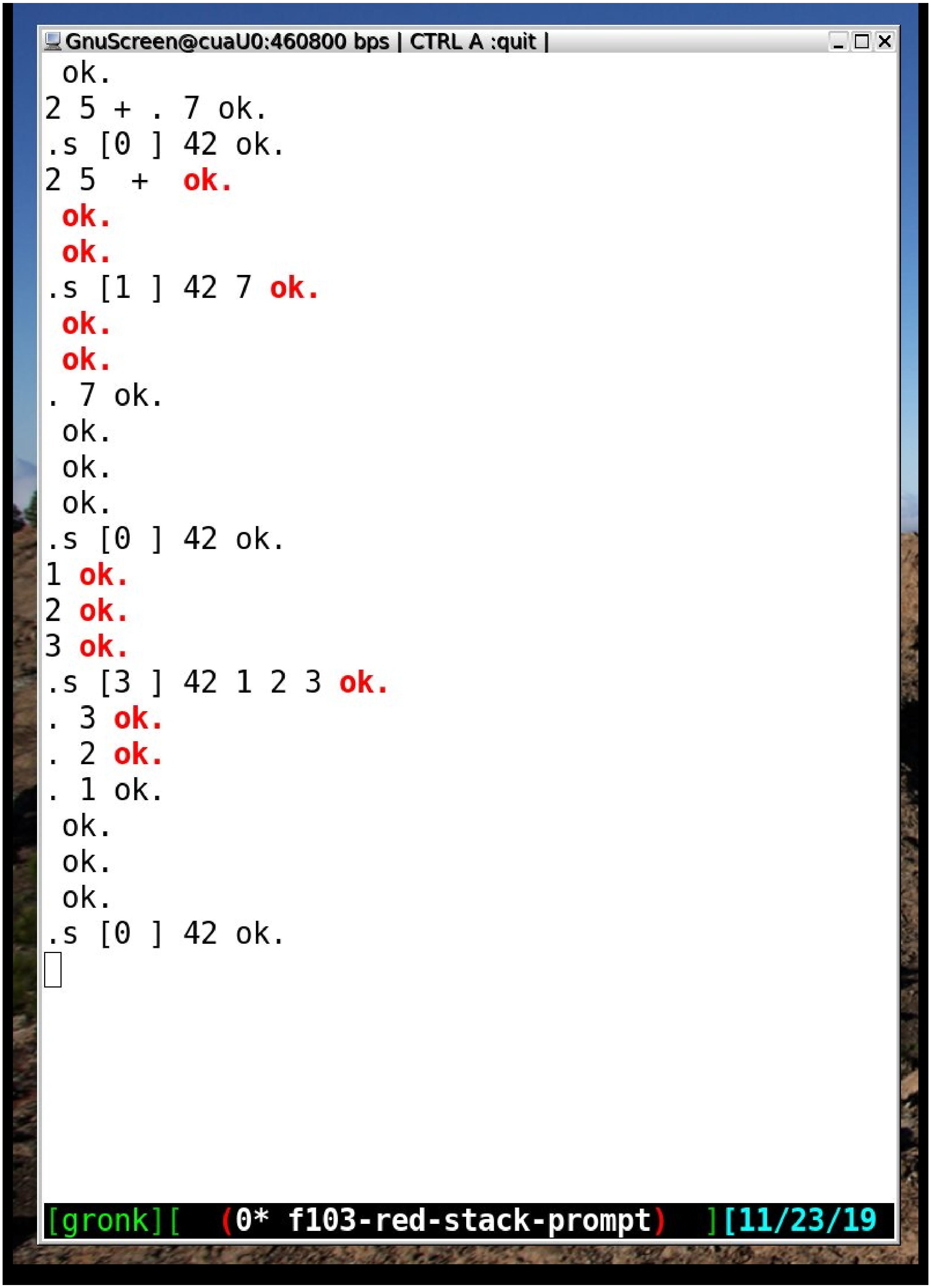
Recently I read a post by Albert Van Der Horst on Comp.Language.Forth (C.L.F) where he suggested colouring the Forth prompt to indicate the Stack status.

What a great idea.

Here is my solution, it is based on code by Glen Worstell; Note that when the stack has NO items, the prompt is the normal colour, but when the stack has one or more items, the prompt is RED.

I think this is a excellent solution as it uses no extra screen real-estate, and red is very eye catching. Who could miss a non empty stack now ?

A stack sensitive prompt



GnuScreen@cuaU0:460800 bps | CTRL A :quit |

```
 ok.
2 5 + . 7 ok.
.s [0 ] 42 ok.
2 5  +  ok.
 ok.
 ok.
.s [1 ] 42 7 ok.
 ok.
 ok.
. 7 ok.
 ok.
 ok.
 ok.
.s [0 ] 42 ok.
1 ok.
2 ok.
3 ok.
.s [3 ] 42 1 2 3 ok.
. 3 ok.
. 2 ok.
. 1 ok.
 ok.
 ok.
 ok.
.s [0 ] 42 ok.
```

```
[gronk][  (0* f103-red-stack-prompt)  ][11/23/19
```

The code

```
\ Program Name: f103-red-stack-prompt.fs
\ Date: Sat 23 Nov 2019 18:23:59 AEDT
\ Copyright 2019 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV2
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F103, Board: Olimex P103
\ Clock: 8 Mhz using the internalRC clock, unless otherwise stated
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\ Requires preloaded support program(s):
\ Standalone: no preloaded support files required
\
\ This Program : make the prompt red when stack >= 1
\
\ ---------------------------------------------------------------------------\
 compiletoram
 \ compiletoflash


 \ designed for a white background with black text, and a red prompt when the
 \ stack is not empty, but you can change to suit from the table below.

 : esc      ( -- ) 27 emit ;
 : black    ( -- cursor colour )  esc ." [30;0m" ;
 : red      ( -- cursor colour )  esc ." [31;1m" ;
 : green    ( -- cursor colour )  esc ." [32;1m" ;
 : yellow   ( -- cursor colour )  esc ." [33;1m" ;
 : blue     ( -- cursor colour )  esc ." [34;1m" ;
 : magenta  ( -- cursor colour )  esc ." [35;1m" ;
 : cyan     ( -- cursor colour )  esc ." [36;1m" ;
 : white    ( -- cursor colour )  esc ." [37;1m" ;

 : prompt ( -- )
   begin
       depth if
       red ." ok. "  black cr    \ prompt
       else ." ok. " cr
```

```
      then
    query  interpret
  again
;

: init.prompt ['] prompt hook-quit ! ;

init.prompt quit
```

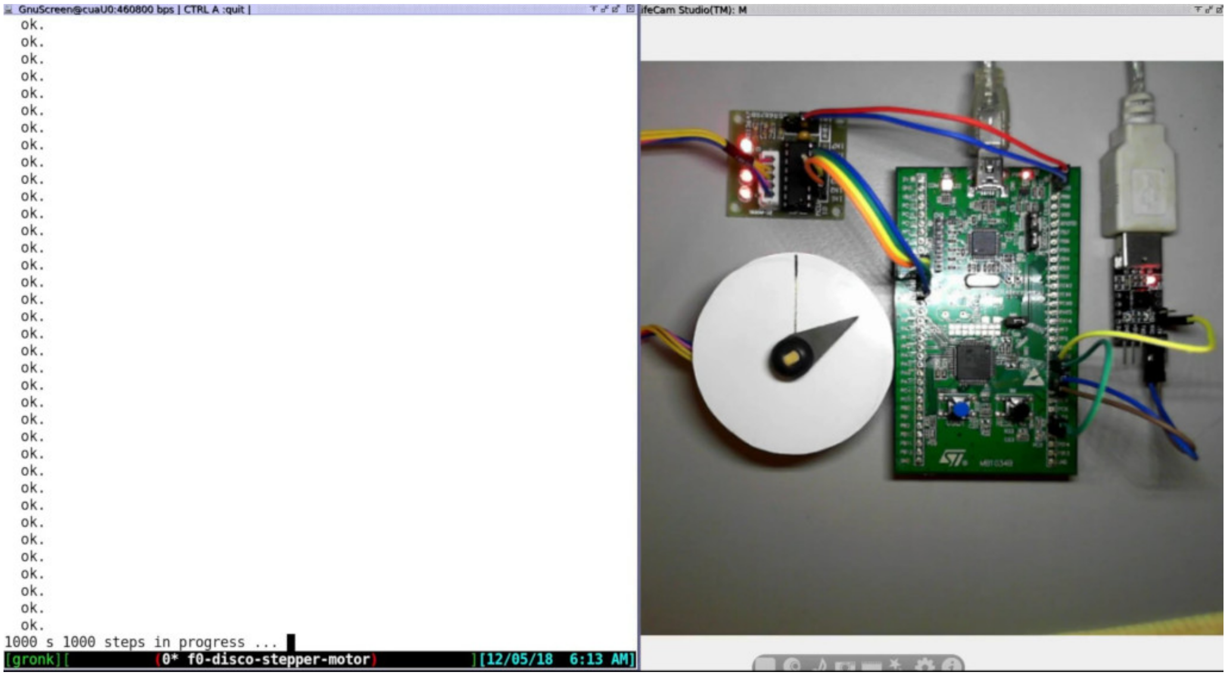
Download: f103-red-stack-prompt.fs

Stepper Motor Demo

A real time Forth Stepper Motor driver demo.

Short Youtube Video

https://www.youtube.com/watch?v=0gGkwmRNnGA


GnuScreen@cuaU0:460800 bps | CTRL A :quit |
ifeCam Studio(TM): M

```
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
  ok.
1000 s 1000 steps in progress ...
[gronk][          (0* f0-disco-stepper-motor)          ][12/05/18  6:13 AM]
```

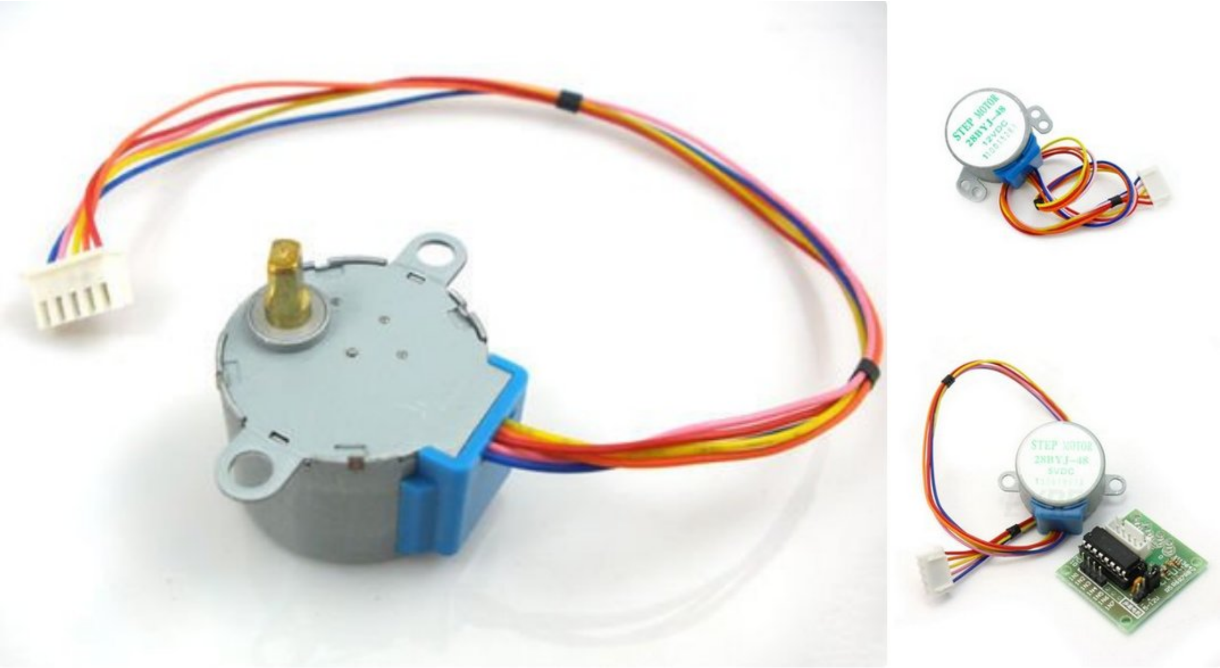
Hearing Impaired

- For the Hearing Impaired (or those that can’t understand my mumbling) **`Download script.txt`** for the video.

Parts Used

- A STM32F0 ‘Discovery Board’
- USB/3.3v dongle

- Stepper motor and driver board: https://www.instructables.com/id/BYJ48-Stepper-Motor/



Fourth Source Code

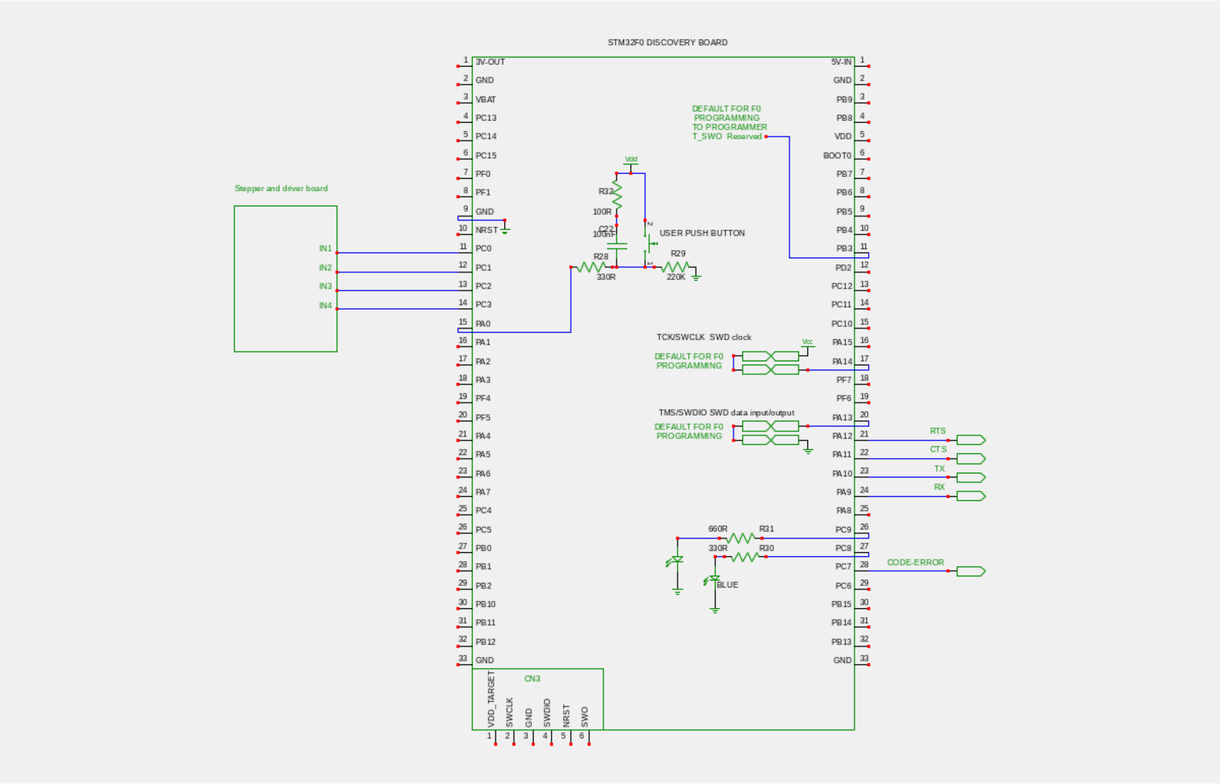
```
\ Program Name: f0-disco-stepper-motor.fs
\ This program may require other support files listed in preload.sh which need to be loaded first
\ Date: Sat 1 Dec 2018 13:18:58 AEDT
\ Copyright 2018  t.porter <terry@tjporter.com.au>, licensed under the GPL
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F051, Board: STM32F0 Discovery Board
\ Clock: 8 Mhz using the internal STM32F051 RC clock, unless otherwise stated
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\
\ This Program : Is a stepper motor controller. Controlled from the terminal "100 steps" result in 100 clockwise steps,
\ "-100 steps" in 100 anticlockwise steps. Program to have a
```

```
absolute-steps counter
\
\
http://forth.org/library/eforth_SOC/eforth_SOC_source/eForth1
/68eforth.zip
\  Stepper: GPIOC: bits 0 - 3
\
\  28-BYJ48 Stepper motor.
\
\  Forward Steps
\  STEP PH4   PH3   PH2   PH1
\  1    OFF   OFF   OFF   ON
\  2    OFF   OFF   ON    OFF
\  3    OFF   ON    OFF   OFF
\  4    ON    OFF   OFF   OFF
\
\  Reverse Steps
\  STEP PH4   PH3   PH2   PH1
\  1    ON    OFF   OFF   OFF
\  2    OFF   ON    OFF   OFF
\  3    OFF   OFF   ON    OFF
\  4    OFF   OFF   OFF   ON
\
\
\ -------------------------------------------------------------
----------------\

 1 constant ph1
 2 constant ph2
 4 constant ph3
 8 constant ph4
 $F constant phase-mask
 0 variable steps
 0 variable absolute-steps
 0 variable counter
 1 variable flag
 %0001 variable motor

 : init.stepper        \ set pins as outputs
   OUTPUT 6  lshift $48000800  bis!      \ GPIOC_MODER_MODER3
   OUTPUT 4  lshift $48000800  bis!      \ GPIOC_MODER_MODER2
   OUTPUT 2  lshift $48000800  bis!      \ GPIOC_MODER_MODER1
   OUTPUT 0  lshift $48000800  bis!      \ GPIOC_MODER_MODER0
   %0001 phase-mask and GPIOC_ODR !
 ;
 : zero ( 0 -- ) 0 absolute-steps ! ." step position counter
reset " cr ;    \ Zero the absolute-steps counter when the
pointer is aligned to the black line on the white dial


 : delay 0 do pause loop                 \ delay to give the
stepper coils enough time to reach peak current
 ;


 : motor-driver ( phase pattern -- )     \ sends the stepper
phase pattern to GPIOC:0-3 without intefering with other bits
   phase-mask and GPIOC_ODR !            \ mask the stepper
drive value with the phase-mask then write to GPIOC
   1000 delay                            \ delay by 1000
clock cycles, more or less. It's only arbitary, basically if
the delay is too short the stepper won't move.
 ;                                       \ if it's too long,
the stepper will rotate too slowly


 : forward-one-step ( -- )               \ motor ONE step
forward
  motor @ ph4 = if ph1 motor !           \ if motor is at
phase 4 already, next step is phase 1
     else  motor @ shl motor !           \ shift the pattern
1 left
  then
  motor @ motor-driver                   \ send the new
pattern to the stepper
;


: reverse-one-step ( -- )                \ motor ONE step
backward-one-step
   motor @ ph1 = if ph4 motor !          \ if motor is at
phase 1 already, next step is phase 4
     else  motor @ shr motor !           \ shift the pattern
1 right
  then
  motor @ motor-driver                   \ send the new
pattern to the stepper
;
```

```
: forward ( steps -- )                   \ forward required
steps
  begin
  pause
  forward-one-step
  1 - dup 0 > while                      \ one forward step
completed, subtract it and test if all steps are done
  repeat                                 \ all steps not
done, go back to begin
  drop                                   \ all steps done,
drop the duplicate
;


: reverse ( steps -- )                   \ reverse required
steps
  begin
  pause
  reverse-one-step
  1 - dup 0 > while                      \ one reverse step
completed, subtract it and test if all steps are done
  repeat                                 \ all steps not
done, go back to begin
  drop                                   \ all steps done,
drop the duplicate
 ;


 : s ( steps -- )                                          \
Step controller Word, determines if steps will be forward or
reverse
   >r r@ . ." steps in progress ... "                      \
Use the Return Stack as a local variable
   r@ 0 > if                                               \
Go UP-MODE
      absolute-steps @ r@ + absolute-steps !               \
update absolute step count
      r@ forward                                           \
send required steps to direction control
        else

           r@ 0 < if                                       \
Go DOWN-MODE
```

```
              absolute-steps @ r@ + absolute-steps !       \
update absolute step count
              r@ abs reverse                               \
convert negative steps to absolute, send to direction control
   then then
   r> drop
\ balance Return Stack
   ."  steps completed, absolute steps: " absolute-steps @ .
cr  \ print absolute step count
     pause
 ;

 : r 2048 * s ;     \ Rotate X complete rotations (2048 steps
per rotation)



\ ------------------ below here only used with
multasking.fs, not needed otherwise -------------------------
----\
\ ------ If not using multitasking.fs, delete/comment-out all
occurrences of the Word "pause" in this file -------\

 : main
   pause                              \ essential Word
 ;


 : init-general                               \ essential
multitasking and initialization Word
   f0-disco-48mhz
   init.stepper
 ;
```

Schematic


STM32F0 DISCOVERY BOARD
Stepper and driver board
IN1
IN2
IN3
IN4
3V-OUT
GND
VBAT
PC13
PC14
PC15
PF0
PF1
GND
NRST
PC0
PC1
PC2
PC3
PA0
PA1
PA2
PA3
PF4
PF5
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PB10
PB11
PB12
GND
5V-IN
GND
PB9
PB8
VDD
BOOT0
PB7
PB6
PB5
PB4
PB3
PD2
PC12
PC11
PC10
PA15
PA14
PF7
PF6
PA13
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
GND
DEFAULT FOR F0 PROGRAMMING TO PROGRAMMER T_SWO Reserved
Vdd
R32
100R
C22
100nF
USER PUSH BUTTON
R28
330R
R29
220K
TCK/SWCLK SWD clock
DEFAULT FOR F0 PROGRAMMING
Vcc
TMS/SWDIO SWD data input/output
DEFAULT FOR F0 PROGRAMMING
RTS
CTS
TX
RX
660R
R31
330R
R30
BLUE
CODE-ERROR
CN3
VDD_TARGET
SWCLK
GND
SWDIO
NRST
SWO

Sophisticated Stepper Driver

This *much more sophisticated* stepper driver example with comprehensive readme and comments by Andrew Palm may be found in the Mecrisp-Stellaris Distribution, STM32F051 directory.

Andrews stepper driver uses a cheap mechanical rotary encoder to control the stepper rotation and vary the brightness of a LED, depending on a push button selectable mode.

As it happens, we both use exactly the same stepper motor and driver board, which being low power is a good choice for USB powered projects.

Readme

This folder contains a set of examples using Mecrisp-Stellaris Forth by Matthias Koch. They were written for an STM32F051K8T6 on a DIP conversion PCB and components on a solderless breadboard. They are listed here in approximately increasing order of complexity.

These examples were adapted from exercises developed while learning C programming for MSP430 and STM32F1/F0 chips using two primary sources:

- "MSP430 State Machine Programming" by Tom Baugh, 2008
- "Discovering the STM32 Microcontroller" by Geoffery Brown, 1992

Andrew Palm
February 2018

---

example-delay.fs - This is a relatively simple "blinky" example. However, it introduces code to set the system clock to 48 MHz using the PLL and internal 8 MHz RC oscillator. Blocking delays are used for an approximate 1 Hz blink rate, with the delay count values derived empirically. Also introduced is the "init" word, which executes on a chip reset. The main program is set up inside this word's definition. In order to make the interpreter available after the program is started with a reset, the bottom of the main loop has the "key? until" construct so that pushing any key on the serial terminal will halt the loop. A "reset" command will restart the program.

----

example-pb-poll.fs - This example is a significant jump up in complexity. It uses the SysTick counter and interrupt to implement a main loop which repeats every 1 millisecond. This provides a time base for program tasks so that blocking delays are not needed in the main loop. Also introduced are "drivers" for a pushbutton and rotary encoder that control two leds. The tasks which directly control the leds "register" with these drivers so that the desired control happens. Why introduce this complication? The reason is that in more complicated programs the functions of the pushbutton and rotary encoder can be changed "on the fly" by registering different "event sinks" or "actions" with them. This is especially useful when implementing menus. (See example-servo.fs below.)

The pushbutton driver allows for two actions, one for a short push and one for a long (> 1 sec) push. In this example a short push turns on led2 and a long push turns it off. It also does software debouncing. The encoder driver assumes that its input connections are debounced in hardware and has cw and ccw actions. In this example a cw turn of the encoder increases the flashing led1 duty cycle, and a ccw turn decreases the duty cycle.

The comment header in the file gives details on the hardware connections. It also contains comments on making the hardware setup stand-alone. Because of the "key? until" construct that allows escape from the main loop, if the terminal is disconnected, the program halts. To defeat this,

```
the USART RX connection on PA10 can be held at 3.3 volts by a
jumper.

-------------------------------------------------------------
--------------
example-pb-int.fs - This is identical in hardware to the
previous example,
but the pushbutton now uses an external interrupt rather than
polling.
This introduces code which sets up an external interrupt and
sets its
priority.  This example also sets the priority of the SysTick
interrupt.

Note that control of the second led by the pushbutton now
runs completely
in the background, as the main loop does not contain a task
for it.

-------------------------------------------------------------
--------------
example-servo.fs - This example adds a third led and a servo
motor.  It
shows how to set up a timer to produce a pwm output, in this
case to
control the servo motor position.  (The servo motor is very
small to avoid
current draw beyond the breadboard power supply.)  The
example also shows
the advantage of using the pushbutton and rotary encoder
driver
registration, as described above, by having two "control
modes."

When the pushbutton is given a long push the control mode is
toggled.
In "Mode 0" the encoder increases and decreases the duty
cycle of the
flashing led, and the pushbutton toggles the second led on
and off with a
short push.  In "Mode 1" the encoder controls the position of
the servo
motor, and a short push on the pushbutton centers the servo
position.  The
third led serves as a mode indicator, off in Mode 0 and on in
```

Mode 1.
The driver registrations for the two control modes are made using the task cntl-mode, which is registered with the pushbutton long push.

---

example-stepper.fs - This example is similar to the servo motor example, except that a 28BYJ-48 5V stepper motor with gear reduction and a ULN2003A darlington driver board is used instead (see eBay). Another difference is when in "Mode 1" the pushbutton simply stops the motor in its current position rather than returning it to a neutral position. The incremental motor steps are done every 5 ms (five main loop ticks), and half-stepping is used.

---

example-adc.fs - This example uses a third led and a 10 Kohm potentiometer in the hardware. The wiper of the pot is input to the ADC, which does a conversion every 10 ms, triggered by a timer. The third led is turned on when the pot wiper voltage is above one half the supply voltage.

The control of the third led using the adc, like the pushbutton control of the second led, is completely in the background.

---

example-ultrasonic.fs - This example uses two timers to trigger and measure the returned echo pulse from an ultrasonic distance sensor. The timer setup code is more complicated than in the previous example, esp. since the time processing the returned echo pulse must use two capture and compare channesl to measure the pulse width. Although the sensor is triggered every 100 ms, the calculation and distance display task is run every 200 ms in the main loop to keep the terminal output reasonable. One could change the trigger period and do the calculation and display in the background.

---

example-nunchuk-i2c.fs - This example uses I2C to communicate with a Wii nunchuk, which contains two pushbuttons, a joystick, and an accelerometer. The raw data from the nunchuk are read over I2C and then converted to values for display on the serial terminal. The I2C drivers can be configured for 100 kHz or 400 kHz communication and can send or receive up to 255 bytes with one call. Error messages can be uncommented to help debug problems.

Another feature introduced with this example is the use of state machines to break up tasks into smaller parts. The reason for this is that a task called from the main loop may take so long to execute that it could not be completed within the time allowed for a single loop, esp. when other tasks must also be completed in the loop time (in this case 1 ms). This simple version of a state machine uses the forth "case" structure. On each call of the task, one of the "of ... endof" pieces is executed. A "state variable" value controls which piece is executed. State machines are used for the tasks nchk-read and nchk-display.

```
--------------------------------------------------------------
--------------
example-ssd1306.fs - Here we use I2C to write to a small
32x128 OLED
display which is used as a 4 row by 21 column character
display.  The
drivers write text strings to the display in either non-
scrolling or
scrolling mode.  This example displays messages in scrolling
mode when the
leds are changed by the pushbutton or rotary encoder.  The
task which
draws the characters on the display is called in the main
loop and uses a
state machine which loops through the 84 character positions,
drawing one
character per loop.  With loop initialization, the display is
updated
completely every 85 ms.

** Note: This example uses a second file named glcdfont.fs.
There is an
         #include directive in the main file to pull the
glcdfont.fs code
         into the main file when it is downloaded via the
serial terminal.
         This directive is for the e4thcom program.  If you
do not use this
         particular terminal program, you can cut and paste
glcdfont.fs
         into example-ssd1306.fs at the place where the
directive is, or
         modify the include directive for your setup.

--------------------------------------------------------------
--------------
example-st7735.fs - This is very similar to example-
ssd1306.fs in
operation, but it uses an Adafruit 160x128 TFT display module
connected
by SPI instead of I2C.  The display is color and graphical,
but it is used
as a monochrome character display with 12 rows and 26 columns
with a blue
background and yellow characters.  (The font and background
```

colors are easy to change.) The SPI1 drivers in this example can be used for other devices, such as sensors.

Makefile - This is a script for the program make, which is usually used to organize the compiling and linking of programs. Here it is a convenient way to invoke terminal commands to flash or bootload the forth kernel to the chip and to connect a serial terminal emulator program to the chip once forth is running.

Final comments. You can create your own examples by mix-and-matching parts of these examples. For example, the servo motor example could be modified to include the ssd1306 display. The display could show which control mode has been selected and show messages when a particular control action is made.

Due to the designed-in upward compatibility of STM32 mcu's, these examples should be easily adapted for larger chips in the STM32F0 family, and for the smaller STM32F030K6T6, which has an identical pinout to the K8T6. Do note, however, that interrupt names should be checked for the forth kernel you are using.

If you use an external 8 MHz crystal instead of the internal oscillator, the 48MHz code must be changed. Modifications would also be necessary for a Nucleo board whose clock is driven by an external source.

Example-stepper.fs

```
\
================================================================
============
\  Program Name: example-stepper.fs for Mecrisp-Stellaris by
Matthias Koch
\
\  This program is a template for larger projects, as it
includes a main
\  loop controlled by the SysTick timer.  There are these
tasks:
\
\     - A flashing yellow led with variable duty cycle
controlled by a
\       rotary encoder.
\     - A green led that is toggled by a short push on a
pushbutton
\     - A red led that indicates the control mode (see below)
\     - A pushbutton driver (interrupt version)
\     - A driver for a rotary encoder with alternate detents
\     - A small stepper motor which is also controlled by the
pushbutton
\       and rotary encoder
\     - When a long (> 1 sec) push is made on the pushbutton,
the control
\       toggles between two modes:
\         Mode 0: Red led off
\                 The pushbutton toggles the green led on and
off
\                 The rotary encoder controls the duty cycle
of the yellow
\                 led
\         Mode 1: Red led on
\                 The pushbutton immediately stops the motor
\                 The rotary encoder controls the stepper
motor
\
\  Hardware:
\     - STM32F051K8T6 on DIP conversion board
\     - Two leds, yellow on PF0, green on PF1 (330 ohm
dropping resistors)
\     - A red led on PA6 (330 ohm dropping resistor)
\     - Pushbutton with internal pullup and software debounce
```

```
on PA4
\     - Rotary encoder with 30 alternate detents on PA0 and
PA1 with
\       internal pullups and hardware debounce.
\     - Small 28BYJ-48 5V stepper motor driven by ULN2003A
darlington
\       driver (from eBay) with these connections:
\
\       Driver board      STM32
\           Ch1            PA8
\           Ch2            PA12
\           Ch3            PA11
\           Ch4            PA15
\            V-            Grnd
\            V+            5V from USB-to-serial board
\
\       Note:  These connections are so that the direction of
rotation
\              facing the motor is the same direction as the
rotary
\              encoder direction.  To reverse this, reverse
the order of
\              the channel connections above.
\
\ -----------------------------------------------------------
--------------
\  To run stand-alone:
\     - This example uses the "init" word so that the main
program will
\       run on a reset by the reset button or a terminal
command.
\     - The main loop uses "key? until" so that when running
with a
\       terminal connected (PA9 TX and PA10 RX) a key press
can break
\       execution.
\     - However, if the terminal RX connection on PA10 is
removed, the
\       program hangs due to the use of key? in the main
loop.
\     - To allow the program to run without the terminal but
retain the
\       use of key? for possible later updating or debugging,
connect PA10
\       to 3.3V (VDD) when not using the terminal (or just
using TX on PA9).
\
\ ---------------------------------------------------------------
--------------
\  Style:
\     - All words are in lower case with parts separted by
hyphens.
\     - Constants are in upper case, with parts separated by
\       underscores.
\     - Variables and words associated with a task are
prefixed by the
\       task name (which is in lower case) with subsequent
parts of
\       variable names having the initial letter in upper
case and
\       separated by hyphens.
\     - Variables acting as function pointers (containing the
address
\       of a word) have a tick prefix.
\ ---------------------------------------------------------------
--------------
\  Andrew Palm
\  2018.02.08
\
==============================================================
============

compiletoflash
\ ---------------------------------------------------------------
--------------
\  Core and peripheral registers
\ ---------------------------------------------------------------
--------------
$E000E010 constant SysTick
  SysTick $00 + constant SysTick_CTRL
  SysTick $04 + constant SysTick_LOAD
  SysTick $08 + constant SysTick_VAL
  SysTick $0C + constant SysTick_CALIB

$E000E100 constant NVIC
  NVIC $000 + constant NVIC_ISER
  NVIC $180 + constant NVIC_ICPR
  NVIC $300 + constant NVIC_IPR0

$E000ED00 constant SCB
```

```
  SCB $01C + constant SCB_SHPR2

$40021000 constant RCC
  RCC $00 + constant RCC_CR
  RCC $04 + constant RCC_CFGR
  RCC $14 + constant RCC_AHBENR
  RCC $18 + constant RCC_APB2ENR
  RCC $1C + constant RCC_APB1ENR

$40022000 constant FLASH
  FLASH $0 + constant FLASH_ACR

$40010000 constant SYSCFG
  SYSCFG $0C + constant SYSCFG_EXTICR2

$40010400 constant EXTI
  EXTI $00 + constant EXTI_IMR
  EXTI $04 + constant EXTI_EMR
  EXTI $0C + constant EXTI_FTSR
  EXTI $14 + constant EXTI_PR

$48000000 constant GPIOA
  GPIOA $00 + constant GPIOA_MODER
  GPIOA $04 + constant GPIOA_OTYPER
  GPIOA $08 + constant GPIOA_OSPEEDR
  GPIOA $0C + constant GPIOA_PUPDR
  GPIOA $10 + constant GPIOA_IDR
  GPIOA $18 + constant GPIOA_BSRR
  GPIOA $20 + constant GPIOA_AFRL
  GPIOA $28 + constant GPIOA_BRR

$48001400 constant GPIOF
  GPIOF $00 + constant GPIOF_MODER
  GPIOF $14 + constant GPIOF_ODR
  GPIOF $18 + constant GPIOF_BSRR
  GPIOF $28 + constant GPIOF_BRR

$40013800 constant USART1
  USART1 $C + constant USART1_BRR

$40002000 constant TIM14
  TIM14 $00 + constant TIM14_CR1
  TIM14 $14 + constant TIM14_EGR
  TIM14 $18 + constant TIM14_CCMR1
  TIM14 $20 + constant TIM14_CCER
  TIM14 $28 + constant TIM14_PSC
  TIM14 $2C + constant TIM14_ARR
  TIM14 $34 + constant TIM14_CCR1

\ ------------------------------------------------------------
--------------
\  Misc. helper words, constants, and variables
\ ------------------------------------------------------------
--------------
\  Null (event sink) action used by pushbutton and rotary
encoder tasks.
: null-act ( -- )  nop ;


\ Wait while the word test with address 'test results in a
true, or until
\ timeout ends.  Returns status = 0 if exits before timeout,
returns
\ status = 1 if timeout occurs.
\ : timed ( 'test -- status )
    \ $FFFF                             \ Timeout value
    \ begin swap dup execute while
      \ swap 1- dup 0= if
        \ 2drop 1 exit                  \ Timed out
      \ then
    \ repeat
    \ 2drop 0                           \ 'test became false
before timeout
\ ;

\ Blocking delays (approximate) for 48 MHz system clock
\ 52000 constant LOOPS/MS
\ : ms-delay ( n -- )   LOOPS/MS * 4 rshift 0 DO nop LOOP ;

52 constant LOOPS/US
: us-delay ( n -- )   LOOPS/US * 4 rshift 0 DO nop LOOP ;

\ Aids to simple printing
\ : hex.dec ( n|u -- )  hex . decimal ;
\ : tab ( -- )  9 emit ;

: h><, ( hx -- )
  dup $00FF and 8 lshift
  swap $FF00 and 8 rshift
  or h,
;
\ -----------------------------------------------------------
--------------
\  Clock setup and timing plus USART1 baud set to 115200 for
cortex-m0.
\  Uses 8 MHz internal oscillator divided by 2 and x12 PLL
multiplier.
\
\  Clock initialization code is modified from usb-f1.txt by
Jean-Claude
\  Wippler.
\  Adjusted for STM32F051 @ 48 MHz (original STM32F100 by
Igor de om1zz).
\ -----------------------------------------------------------
--------------
: 48MHz ( -- )
\ Set the main clock to 48 MHz, keep baud rate at 115200

  %1 RCC_CR bis!                        \ set HSION
  begin %1 1 lshift RCC_CR bit@ until  \ wait for HSIRDY

  %10001 FLASH_ACR !    \ One flash wait state, enable
prefetch buffer

  \ HSI clock /2 = 4 MHz source for PLL
  %1111 18 lshift RCC_CFGR bic!
  %1010 18 lshift RCC_CFGR bis!   \ PLL factor: 4 MHz * 12 =
48 MHz

  %1111 4 lshift RCC_CFGR bic!    \ HPRE DIV 1, HCLK = SYSCLK
  %111  8 lshift RCC_CFGR bic!    \ PPRE DIV 1, PCLK = HCLK

  %1 24 lshift RCC_CR bis!                \ set PLLON
  begin %1 25 lshift RCC_CR bit@ until    \ wait for PLLRDY

  %11 RCC_CFGR bic!
  %10 RCC_CFGR bis!               \ PLL is system clock

  $1A1 USART1_BRR !               \ Set console baud rate to
115200
;

\ -----------------------------------------------------------
--------------
\  Set up SysTick for main loop 1 ms tick
\ -----------------------------------------------------------
--------------
1 constant MAIN_TICKS  \ Number of systicks for main loop
repeat
MAIN_TICKS variable main-Tick-Cnt

1 constant SysTick_PRIORITY
-1 constant SysTick_IRQn

: systick-handler ( -- )
  main-Tick-Cnt @ if
    -1 main-Tick-Cnt +!
  then
;

: wait-for-tick ( -- )
  begin main-Tick-Cnt @ 0= until
  MAIN_TICKS main-Tick-Cnt !
;

: systick-init ( -- )
  %011 SysTick_CTRL bic!    \ Disable timer and interrupt
  48000 1 - SysTick_LOAD !  \ (SystemCoreClock/1000)-1 for 1
ms tick
  0 SysTick_VAL !
  %101 SysTick_CTRL bis!    \ Set CLKSOURCE, ENABLE bits

  \ Set SysTick interrupt priority
  SysTick_PRIORITY 6 lshift $FF and  \ Priority byte
  SysTick_IRQn %11 and 3 lshift      \ Bit shift
  lshift
  SysTick_IRQn %1111 and 8 - 2 rshift 2 lshift  \ Byte offset
from SCB_SHPR2
  SCB_SHPR2 + bis!                    \ Write priority bits

  ['] systick-handler irq-systick !   \ Register systick-
handler
  %010 SysTick_CTRL bis!    \ Enable system tick interrupt
(bit TICKINT)
;

\ -----------------------------------------------------------
--------------
\  Initialize gpio peripheral clocks and pins
\ -----------------------------------------------------------
```

```
--------------
7 constant EXTI4_15_IRQn
2 constant EXTI4_15_PRIORITY

: gpio-init ( -- )
  \ Enable GPIOF peripheral clock and set pins PF0 and PF1 as
output
  \ (push-pull, low speed) for leds
  %1 22 lshift RCC_AHBENR bis!
  %0101 GPIOF_MODER !

  \ Enable GPIOA peripheral clock and set pins PA0, PA1, and
PA4 to
  \ input with internal pullups
  %1 17 lshift RCC_AHBENR bis!
  %01 8 lshift %01 2 lshift or %01 or GPIOA_PUPDR bis!

  \ Set up external interrupt on PA4
  %111 SYSCFG_EXTICR2 bic!    \ Assign PA4 to EXTI line
4(reset value)
  %1 4 lshift EXTI_IMR bis!   \ Enable interrupt, line 4
  %1 4 lshift EXTI_FTSR bis!  \ Falling edge trigger enabled,
line 4
  %1 4 lshift EXTI_EMR bis!   \ Enable event, line 4
  \ Set interrupt priority
  EXTI4_15_PRIORITY 6 lshift $FF and  \ Priority byte
  EXTI4_15_IRQn %11 and 3 lshift      \ Bit shift
  lshift
  EXTI4_15_IRQn 2 rshift 2 lshift     \ Byte offset from
NVIC_IPR0
  NVIC_IPR0 + bis!                    \ Write priority bits
  \ Enable interrupt
  %1 EXTI4_15_IRQn lshift NVIC_ISER bis!

  \ Set pin PA6 as output for led (push-pull, low speed)
  %01 12 lshift GPIOA_MODER bis!

  \ Set pins PA8, PA11, PA12, and PA15 as output to stepper
motor
  \ (push-pull, low speed)
  %01 16 lshift %01 22 lshift or %01 24 lshift or %01 30
lshift or
  GPIOA_MODER bis!

;
```

```
\ -----------------------------------------------------------
--------------
\  Pushbutton driver for pushbutton pb0 on pin PA4 with
software debounce.
\  This is an interrupt version that assumes that pb1-task is
called in the
\  main loop.  The interrupt is used to start a timer on a
falling edge.
\  Input pin value is assumed low on push, high (pulled up)
on release.
\  Pushbutton action occurs when button is released.
\  There are two types of pushes, short (<= 1 sec) and long
(> 1 sec).
\  A client task registers short and long push actions (event
sinks)
\  by loading the associated word addresses into the pointer
variables
\  'pb1-Short-Act and 'pb1-Long-Act, resp.
\ -----------------------------------------------------------
--------------
%1 4 lshift constant PB1_MSK

PB1_MSK variable pb1-Val
0 variable pb1-Timer-Cnt
0 variable pb1-Timer-On?

' null-act variable 'pb1-Short-Act
' null-act variable 'pb1-Long-Act

: pb1-get ( -- )  GPIOA_IDR @ PB1_MSK and pb1-Val ! ;

: pb1-handler
  \ If falling edge detected on PA4, start pushbutton timer
  EXTI_PR @ PB1_MSK and if    \ Interrupt for EXTI line 4
asserted?
    1 pb1-Timer-On? !         \ Turn timer on
    PB1_MSK EXTI_PR bis!      \ Reset the interrupt flag
  then
;
: pb1-init ( -- )
  pb1-get
  ['] null-act 'pb1-Short-Act !
  ['] null-act 'pb1-Long-Act !
  ['] pb1-handler irq-exti4_15 !    \ Register systick-
```

```
handler
;

: pb1-task
  pb1-Timer-On? if
    1 pb1-Timer-Cnt +!    \ Increment timer
    pb1-get
    pb1-Val @ PB1_MSK = if    \ Button released?
      pb1-Timer-Cnt @ 1000 > if   \ Long push if more than
1000 main ticks
        'pb1-Long-Act @ execute
      else
        pb1-Timer-Cnt @ 20 > if   \ Debounce delay 20 main
loop ticks
          'pb1-Short-Act @ execute
        then
      then
      0 pb1-Timer-Cnt !   \ Reset timer count
      0 pb1-Timer-On? !   \ Turn timer off
    then
  then
;

\ -----------------------------------------------------------
--------------
\  Rotary encoder driver (alternating detent type) on PA0 and
PA1.
\  (If encoder does not have alternating detents, see comment
below.)
\  If the encoder is mechanical, it is assumed that the
channels are
\  debounced in hardware.
\  This is a polling version that assumes that pb1-task is
called in the
\  main loop.  Input pins are pulled up.
\  A client task registers clockwise and counter-clockwise
actions (event
\  sinks) by loading the associated word addresses into the
pointer
\  variables 'enc1-CW-Act and 'enc1-CCW-Act, resp.
\ -----------------------------------------------------------
--------------
' null-act variable 'enc1-CW-Act
' null-act variable 'enc1-CCW-Act
```

```
%01 constant CH_A_MSK
%10 constant CH_B_MSK

CH_A_MSK variable enc1-ChA-Val
CH_A_MSK variable enc1-ChA-Last-Val
CH_B_MSK variable enc1-ChB-Val

: enc1-cha-get ( -- )  GPIOA_IDR @ CH_A_MSK and enc1-ChA-Val
! ;

: enc1-chb-get ( -- )  GPIOA_IDR @ CH_B_MSK and enc1-ChB-Val
! ;

: enc1-init ( -- )
  enc1-cha-get
  enc1-chb-get
  enc1-ChA-Val @ enc1-ChA-Last-Val !
;

: enc1-task ( -- )
  enc1-cha-get
  enc1-chb-get

  enc1-ChA-Val @ 0= enc1-ChA-Last-Val @ 0<> and if
    enc1-ChB-Val @ 0<> if
      'enc1-CW-Act @ execute
    else
      'enc1-CCW-Act @ execute
    then
  then

  \ If rotary encoder does not have alternating detents,
comment out
  \ the if-then below
  enc1-ChA-Val @ 0<> enc1-ChA-Last-Val @ 0= and if
    enc1-ChB-Val @ 0<> if
      'enc1-CCW-Act @ execute
    else
      'enc1-CW-Act @ execute
    then
  then

  enc1-ChA-Val @ enc1-ChA-Last-Val !
;
```

```
\ -----------------------------------------------------------
--------------
\  Stepper motor tasks
\ -----------------------------------------------------------
--------------
' null-act variable 'stpr-Incr

\ Half-step bit pattern bytes for stepper motor.  Patterns
are stored in
\ lower nybble of each byte, A channel in bit 0, A-bar in bit
1, B in bit
\ 2, B-bar in bit 3.
\ High and low bytes of halfwords are switched for storage by
\ custom word h<>, so array indices act as if they were for a
byte array
\ when accessing it with c@ word.
create A-A/-B-B/
  $0901 h><, $0504 h><, $0602 h><, $0A08 h><,

0 variable stpr-Indx      \ Current bit pattern index

$9900 constant STPR_MSK   \ Bit mask for stepper motor
control pins

: stpr-wrt (  -- )
  \ Write current bit pattern to port control pins
  A-A/-B-B/ stpr-Indx @ + c@     \ Get current bit pattern
  $0000                          \ Initialize port pin mask
  over %0001 and 8 lshift or    \ Channel A bit to PA8
  over %0010 and 10 lshift or   \ Channel A-bar bit to PA11
  over %0100 and 10 lshift or   \ Channel B bit to PA12
  over %1000 and 12 lshift or   \ Channel B-bar bit to PA15
  dup not STPR_MSK and GPIOA_BRR bis!   \ Reset zero bits
  GPIOA_BSRR bis! drop                  \ Set one bits
;

: stpr-cw-incr ( -- )
  \ Move stepper motor one incremental step clockwise
  stpr-Indx @ 1 + 8 mod stpr-Indx !
  stpr-wrt
;

: stpr-ccw-incr ( -- )
  \ Move stepper motor one incremental step counter-clockwise
  stpr-Indx @ 1 - dup 0< if
```

```
    7 stpr-Indx ! drop
  else
    stpr-Indx !
  then
  stpr-wrt
;

\ The number of main loop ticks for one (arbitrary) unit of
rotation
\ corresponding to one encoder detent is 1035.  One of these
units is
\ approximately 18 degrees.  There are 1035/5 = 207
incremental motor steps
\ per one of these units.  These values were chosen for a
particular motor
\ with high gear reduction and need to be changed for other
motors.
1035 constant STPR_TKS/UNIT
5 constant STPR_TKS/STEP

0 variable stpr-Cnt

\ Stop motor in current position (pushbutton client)
: stpr-stop ( -- )
  0 stpr-Cnt !
;

: stpr-task ( -- )
  \ If stpr-Cnt > 0, move motor every STRR_TKS/STEP ticks
  \ in direction determined by function pointer 'stpr-Incr
  stpr-Cnt @ 0 > if
    -1 stpr-Cnt +!
    stpr-Cnt @ STPR_TKS/STEP mod 0= if
      'stpr-Incr @ execute
    then
  then
;

: stpr-cw-step ( -- )
  \ Move motor one step clockwise (client of rotary encoder)
  ['] stpr-ccw-incr 'stpr-Incr @ = if
    ['] stpr-cw-incr 'stpr-Incr !       \ Change direction
    STPR_TKS/UNIT stpr-Cnt !            \ and reset tick
count
  else
```

```
    STPR_TKS/UNIT stpr-Cnt +!   \ Same directon, incr tick
count
  then
;

: stpr-ccw-step ( -- )
  \ Move motor one step counter-clockwise (client of rotary
encoder)
  ['] stpr-cw-incr 'stpr-Incr @ = if
    ['] stpr-ccw-incr 'stpr-Incr !       \ Change direction
    STPR_TKS/UNIT stpr-Cnt !             \ and reset tick
count
  else
    STPR_TKS/UNIT stpr-Cnt +!   \ Same direction, incr tick
count
  then
;

: stpr-init ( -- )
  ['] stpr-cw-incr 'stpr-Incr !
  0 stpr-Cnt !
  0 stpr-Indx !
  stpr-wrt
;

\ -----------------------------------------------------------
--------------
\  Yellow led task (client of rotary encoder task)
\  Assumes that led is on if output pin is high.
\ -----------------------------------------------------------
--------------
%1 constant LED1_MSK

500 constant LED1_DFLT_TKS
1000 constant LED1_MAX_TKS
50 constant LED1_MIN_TKS
50 constant LED1_DEL_TKS

LED1_DFLT_TKS variable led1-On-Cnt
LED1_DFLT_TKS variable led1-Off-Cnt
LED1_DFLT_TKS variable led1-Cnt

: led1-dc+ ( -- )
  \ Increase led duty cycle
  led1-On-Cnt @ LED1_DEL_TKS + dup dup LED1_MAX_TKS < if
```

```
    led1-On-Cnt !
    LED1_MAX_TKS swap - led1-Off-Cnt !
  else
    drop drop
  then
;

: led1-dc- ( -- )
  \ Decrease led duty cycle
  led1-On-Cnt @ LED1_DEL_TKS - dup dup LED1_MIN_TKS >= if
    led1-On-Cnt !
    LED1_MAX_TKS swap - led1-Off-Cnt !
  else
    drop drop
  then
;

: led1-init ( -- )
  LED1_DFLT_TKS led1-On-Cnt !
  LED1_DFLT_TKS led1-Off-Cnt !
  LED1_DFLT_TKS led1-Cnt !
;

: led1-task ( -- )
  -1 led1-Cnt +!
  led1-Cnt @ 0= if
    GPIOF_ODR @ LED1_MSK and dup if   \ Is led on?
      GPIOF_BRR bis!                  \ Turn led off
      led1-Off-Cnt @ led1-Cnt !       \ Load off tick count
    else
      drop LED1_MSK GPIOF_BSRR bis!   \ Turn led on
      led1-On-Cnt @ led1-Cnt !        \ Load on tick count
    then
  then
;

\ -----------------------------------------------------------
--------------
\  Green led task (client of pushbutton task)
\  Assumes that led is on if output pin is high.
\ -----------------------------------------------------------
--------------
%10 constant LED2_MSK

: led2-off ( -- )  LED2_MSK GPIOF_BRR bis! ;
: led2-on ( -- )  LED2_MSK GPIOF_BSRR bis! ;

: led2-tggl ( -- ) LED2_MSK GPIOF_ODR xor! ;

: led2-init ( -- )
  led2-off
;

\ ----------------------------------------------------------
--------------
\  Red led task (client of control task)
\  Assumes that led is on if output pin is high.
\ ----------------------------------------------------------
--------------
%1 6 lshift constant LED3_MSK

: led3-off ( -- )  LED3_MSK GPIOA_BRR bis! ;

: led3-on ( -- )  LED3_MSK GPIOA_BSRR bis! ;

\ ----------------------------------------------------------
--------------
\  Control task
\  Task toggles the control mode when there is a long push on
the
\  pushbutton.
\ ----------------------------------------------------------
--------------
1 variable cntl-Next-Mode  \ Control mode state variable

: cntl-mode-0 ( -- )
  led3-off
  ['] led1-dc+ 'enc1-CW-Act !
  ['] led1-dc- 'enc1-CCW-Act !
  ['] led2-tggl 'pb1-Short-Act !
;

: cntl-mode-1 ( -- )
  led3-on
  ['] stpr-cw-step 'enc1-CW-Act !
  ['] stpr-ccw-step 'enc1-CCW-Act !
  ['] stpr-stop 'pb1-Short-Act !
;
```

```
: cntl-mode ( -- )
  \ Set function pointers for new mode, set next mode value
  cntl-Next-Mode @ case
    0 of
      cntl-mode-0
      1 cntl-Next-Mode !
    endof
    1 of
      cntl-mode-1
      0 cntl-Next-Mode !
    endof
  endcase
;

: cntl-init ( -- )
  \ Register with pushbutton long push
  ['] cntl-mode 'pb1-Long-Act !
  \ Initialize in mode 0
  cntl-mode-0
  1 cntl-Next-Mode !
;

\ -----------------------------------------------------------
--------------
\  Main program that starts on reset
\ -----------------------------------------------------------
--------------
: init ( -- )

  \ ---------------- Initialization -------------------------
--------------
  \ Hardware
  dint          \ Global interrupt disable
  48MHz         \ Set system clock to 48 MHz
  systick-init  \ Set system tick for 1 ms
  gpio-init     \ Initialize port pins and pin interrupts
  eint          \ Global interrupt enable

  \ Tasks
  pb1-init
  enc1-init
  cntl-init
  stpr-init
  led1-init
  led2-init
  \ ---------------- Main loop ------------------------------
--------------
  begin wait-for-tick
    pb1-task
    enc1-task
    led1-task
    stpr-task
  key? until            \ Escape from main loop on key press
;
\ -----------------------------------------------------------
--------------
compiletoram
```

STM32 Book List

> What’s New ? <

STM32 Boards

A sample supported by Mecrisp-Stellaris.

See HERE the full list of Supported Chips and Boards.

STM32F Discovery Boards

STM32F0 Discovery Board

- Price: $15
- Manufacturer: STM Microelectronics
- MCU: STM32F051 up to 48 MHz
- Resources: 64K FLASH, 8K parity checked RAM
- Features, SWD input, SWD programmer, the usual GPIO pins
- Flashing: SWD via the USB connector, see the stlink flashing howto
- STM32F0 Discovery Board URL [http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF253215?sc=internet/evalboard/product/253215.jsp]
- Mecrisp-Stellaris serial connections

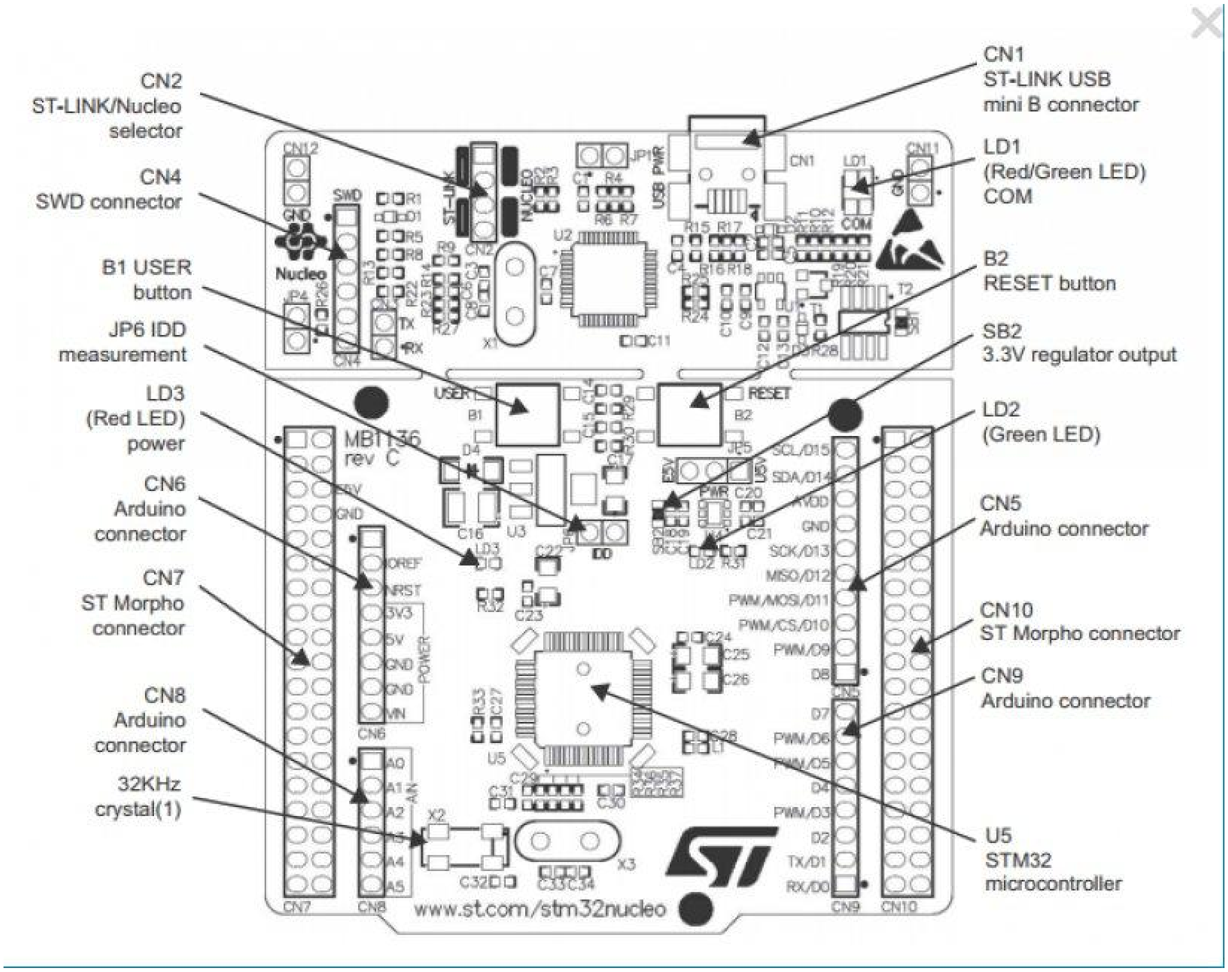
Note

Negatives: The pins on top are short which makes plug on leads unreliable and this board does not come with a RTC xtal.

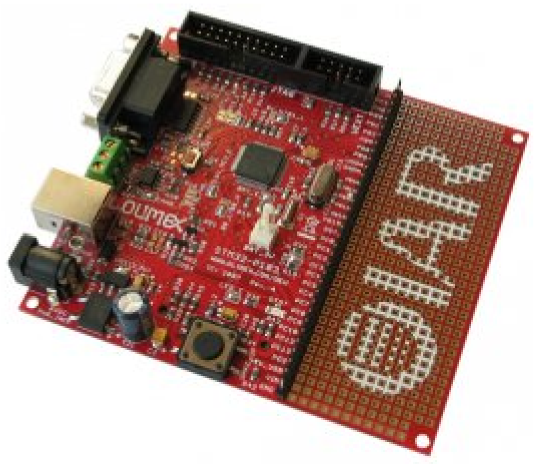
NUCLEO-F411RE Discovery Board

- Price: $17.00
- Manufacturer: STM Microelectronics

- MCU: STM32F411RET6
- Resources: 512 KB Flash, 128 KB SRAM, GPIO (50), USB
- Flashing: SWD via the USB connector, see the stlink flashing howto
- Supports Arduino and ST morpho connectivity
- NUCLEO-F411 RE Discovery Board URL [http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1847/PF260320]
- Mecrisp-Stellaris serial connections


CN2
ST-LINK/Nucleo
selector
CN4
SWD connector
B1 USER
button
JP6 IDD
measurement
LD3
(Red LED)
power
CN6
Arduino
connector
CN7
ST Morpho
connector
CN8
Arduino
connector
32KHz
crystal(1)
CN1
ST-LINK USB
mini B connector
LD1
(Red/Green LED)
COM
B2
RESET button
SB2
3.3V regulator output
LD2
(Green LED)
CN5
Arduino connector
CN10
ST Morpho connector
CN9
Arduino connector
U5
STM32
microcontroller
MB1136
rev C
www.st.com/stm32nucleo

Olimex P103 Board Picture.



Olimex P103 Board

- Nicely made, high quality, new genuine STM32F103RBT6 chip
- MCU STM32F103RBT6 with 128KB Flash and 20KB Ram
- USB, CAN connector, RS232, 2.1mm 6V power socket.
- Flashing: via JTAG connector
- older board, around $50 new 7 years ago.

Notes about the STM32F103C8T6 used in the “Blue Pill” and “Shenzhen-LC”

The STM32F103 is a old ARM Cortex-M3 chip, first announced 11 Jun 2007 and the third ARM family by STMicroelectronics. Whilst this chip is still highly advanced and feature packed compared to all the ancient ATMEL MEGA stuff, later STM MCU’s have more modern peripherals.

Warning

If you’re coming from a Arduino Atmel Mega series the STM32F103 will probably look like advanced alien tech from the future, but don’t be fooled, this is a old chip now.

See also

The STM32F103C8T6 is advertized as having only 64kB Flash, BUT it appears to have a “hidden” second 64kB bank of flash making a total of 128kB Flash! See this page.

STM32F103 Limitations

```
Function                         Limitation
========                         ==========
CAN Bus                          Shares memory with the usb
hardware, which means that usb and canbus cannot be enabled
at THE SAME TIME
Serial Comms                     NO AUTOBAUD
USB                              Shares memory with the
canbus hardware, which means that usb and can cannot be
enabled at THE SAME TIME.
USB                              must use a CRYSTAL unlike
later stm32 chips which use only a internal RC clock for USB
Digital to Analogue Converter    DOESN’T have one
GPIO I/O port registers          Have to be accessed as 32-
bit words, half-word or byte accesses are NOT allowed. Very
CONFUSING to configure.
GPIO I/O port register           Very CONFUSING to configure
because of GPIOx_CRH and GPIOx_CRL. Cortex-M0 is MUCH more
logical making config trivial.
I2C                              There are SIX pages of
ERRATA for the I2C peripheral
DFU                              NO DFU bootloader (for
flashing via USB)
```

The Blue Pill boards all use recycled STM32F103C8T6’s (in my opinion) and I’ve read about Blue Pills coming out with GigaDevices Cortex M3 “GD32F130xx” (64KB Flash MAX), STM32F103 clones which I think would also be recycled e-waste as that chip came out in 2014.

Certainly one can still buy STM32F103C8T6’s brand new from reputable resellers such as AVENET, but they’re more expensive than the later and more capable STM models because the STM32F103C8T6’s are a old chip and I guess STM are just keeping the production line going for spare parts.

My personal opinion is that the STM32F103C8T6 is not such a great choice for new projects, but if you have one there is now a Blue Pill Forth Starter Kit for it.

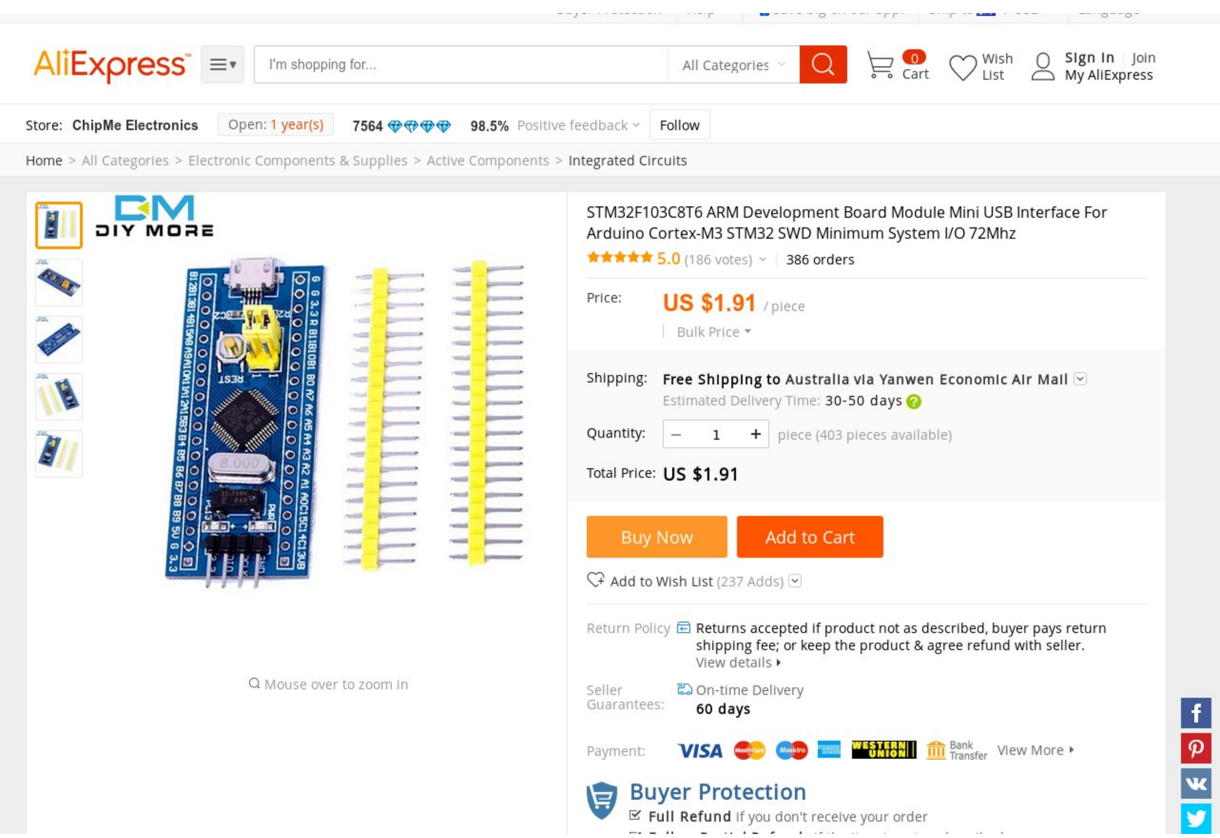
Very Cheap Chinese STM32F103C8T6 Boards

STM32F103C8T6 ‘Blue Pill’

As of 2019 the STM32F103 is no longer protected under mask work copyright, and new clones seem to be appearing.

Specs

- Price: around $2
- Manufacturer: various
- MCU: STM32F103C8T6 Cortex-M3, probably recycled from E-Waste as new chips are around $5 - $7 each from reputable distributors.
- Resources: 64-128KB Flash, USB, 8MHz crystal
- Flashing: Serial Bootloader, see Mecrisp-Stellaris serial connections and the Serial Bootloader Howto


AliExpress
I'm shopping for...
All Categories
0
Cart
Wish List
Sign In
Join
My AliExpress
DIY MORE
Mouse over to zoom in
Buy Now
Add to Cart
VISA
Bank Transfer
View More

Store: **ChipMe Electronics** Open: 1 year(s) 7564 98.5% Positive feedback Follow

Home > All Categories > Electronic Components & Supplies > Active Components > Integrated Circuits

STM32F103C8T6 ARM Development Board Module Mini USB Interface For Arduino Cortex-M3 STM32 SWD Minimum System I/O 72Mhz

5.0 (186 votes) | 386 orders

Price: US $1.91 / piece

Bulk Price

Shipping: Free Shipping to Australia via Yanwen Economic Air Mail

Estimated Delivery Time: 30-50 days

Quantity: – 1 + piece (403 pieces available)

Total Price: US $1.91

Add to Wish List (237 Adds)

Return Policy: Returns accepted if product not as described, buyer pays return shipping fee; or keep the product & agree refund with seller. View details

Seller Guarantees: On-time Delivery 60 days

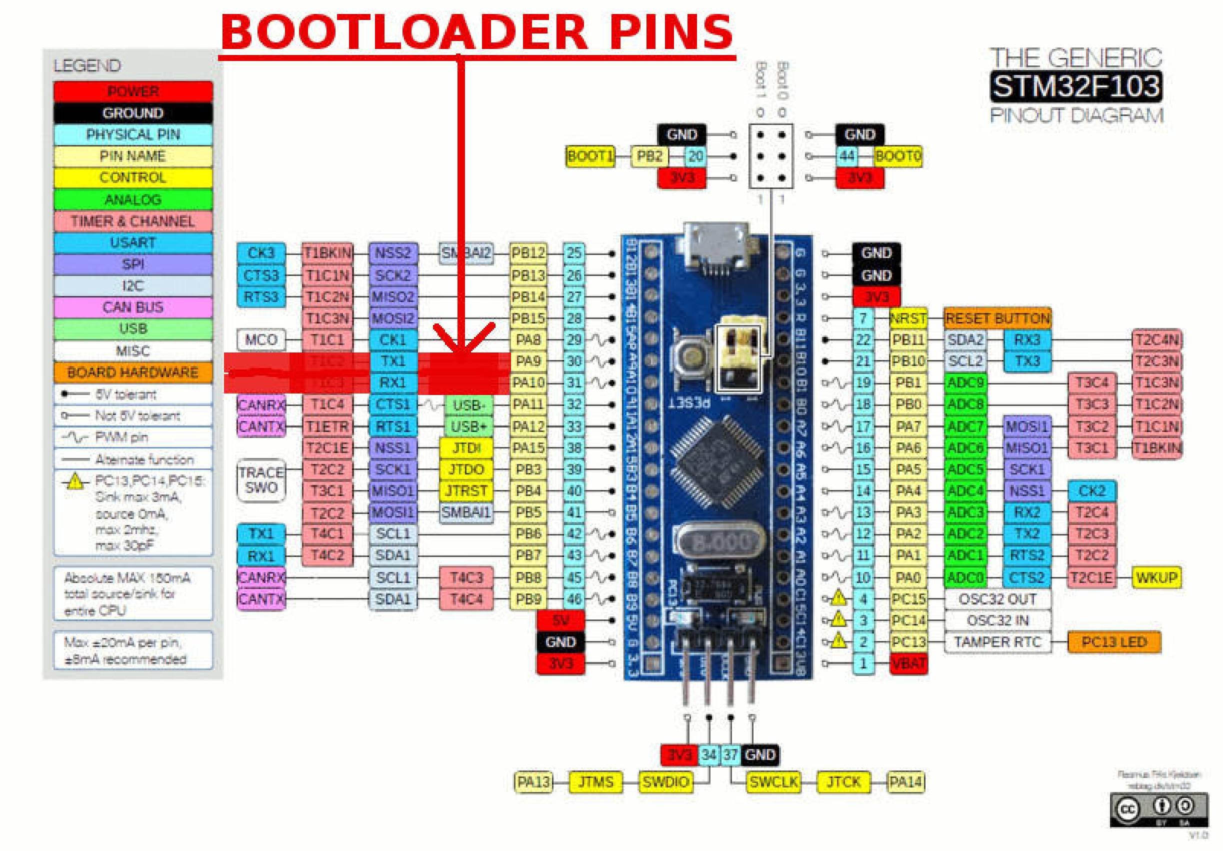
Payment:

Buyer Protection

☑ Full Refund if you don't receive your order

‘Blue Pill’ Pinout Graphic

Author [Rasmus-Friis-Kjeldsen]


BOOTLOADER PINS
THE GENERIC
STM32F103
PINOUT DIAGRAM
LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
Boot 1
Boot 0
GND
BOOT1 PB2 20
3V3
GND
44 BOOT0
3V3
CK3 T1BKIN NSS2 SMBAI2 PB12 25
CTS3 T1C1N SCK2 PB13 26
RTS3 T1C2N MISO2 PB14 27
T1C3N MOSI2 PB15 28
MCO T1C1 CK1 PA8 29
T1C2 TX1 PA9 30
T1C3 RX1 PA10 31
CANRX T1C4 CTS1 USB- PA11 32
CANTX T1ETR RTS1 USB+ PA12 33
T2C1E NSS1 JTDI PA15 38
TRACE SWO T2C2 SCK1 JTDO PB3 39
T3C1 MISO1 JTRST PB4 40
T2C2 MOSI1 SMBAI1 PB5 41
TX1 T4C1 SCL1 PB6 42
RX1 T4C2 SDA1 PB7 43
CANRX SCL1 T4C3 PB8 45
CANTX SDA1 T4C4 PB9 46
5V
GND
3V3
GND
GND
3V3
7 NRST RESET BUTTON
22 PB11 SDA2 RX3 T2C4N
21 PB10 SCL2 TX3 T2C3N
19 PB1 ADC9 T3C4 T1C3N
18 PB0 ADC8 T3C3 T1C2N
17 PA7 ADC7 MOSI1 T3C2 T1C1N
16 PA6 ADC6 MISO1 T3C1 T1BKIN
15 PA5 ADC5 SCK1
14 PA4 ADC4 NSS1 CK2
13 PA3 ADC3 RX2 T2C4
12 PA2 ADC2 TX2 T2C3
11 PA1 ADC1 RTS2 T2C2
10 PA0 ADC0 CTS2 T2C1E WKUP
4 PC15 OSC32 OUT
3 PC14 OSC32 IN
2 PC13 TAMPER RTC PC13 LED
1 VBAT
RESET
8.000
PC13
3V3 34 37 GND
PA13 JTMS SWDIO SWCLK JTCK PA14
V1.0

See this excellent webpage for detailed information: http://wiki.stm32duino.com/index.php?title=Blue_Pill

Warning

A friend bought seven of these recently (2018) from two Amazon vendors, and 100% were faulty. Amazon returned her money and she bought two STM32 Nucleos from Amazon, both worked perfectly.

STM32F103 Clones

Note

The GigaDevices Cortex M3 “GD32F130xx” (64KB Flash MAX), STM32F103 clone has been around since 2014, and are now starting to appear in the BluePill and similar boards. This clone is not 100% identical to the STM32F103 and will have different USB ID numbers etc, so may it not be recognised by your programmer.

Note

The CKS CS32F103C8T6CS32 is yet another clone of the STM32F103 appearing in the Blue Pill. CS32F103C8T6CS32 comes with an Arm Cortex-M3 core clocked 108 MHz frequency with zero Flash wait states! , 64 to 128KB flash, and 20 KB RAM. A datasheet and user manual are available in Chinese http://www.ckscup.com/upload/CS32F103%E6%89%8B%E5%86%8C.pdf.

Forum Blue Pill Problem Posts

Boot0

The batch I received refuses to enter boot-mode when jumping BOOT0. I noticed that when using the BOOT0 jumper, the pin on the chip only gets around 1,1V. Bypassing this and crudely connecting 3.3 directly to the pin makes it work! There is a 100k resistor in the way.

USB not working

Just for the sake of completeness, I swapped one of the STM32’s on a non working board with a Farnell UK sourced part, after which the board worked entirely as expected. Taken together with the different chip ID, it seems certain that the chips are not actually ST parts and so any of these boards may well not work as expected, certainly for the USB, but quite possibly for other peripherals too. It is one thing getting a different chip with appropriate marking, but mislabeled parts make them too big a gamble for me that taints all the STM32 system minimum boards, with it all but impossible to ensure any quality standard at all.

Shenzhen-LC “mini STM32F103C8T6-board”

Note

I bought 12 of these in 2014 for $9 each, one was dead out of the box and missing option jumpers, one died while flashing. At least they appear to have bonafide STM32F103C8T6 MCUS.

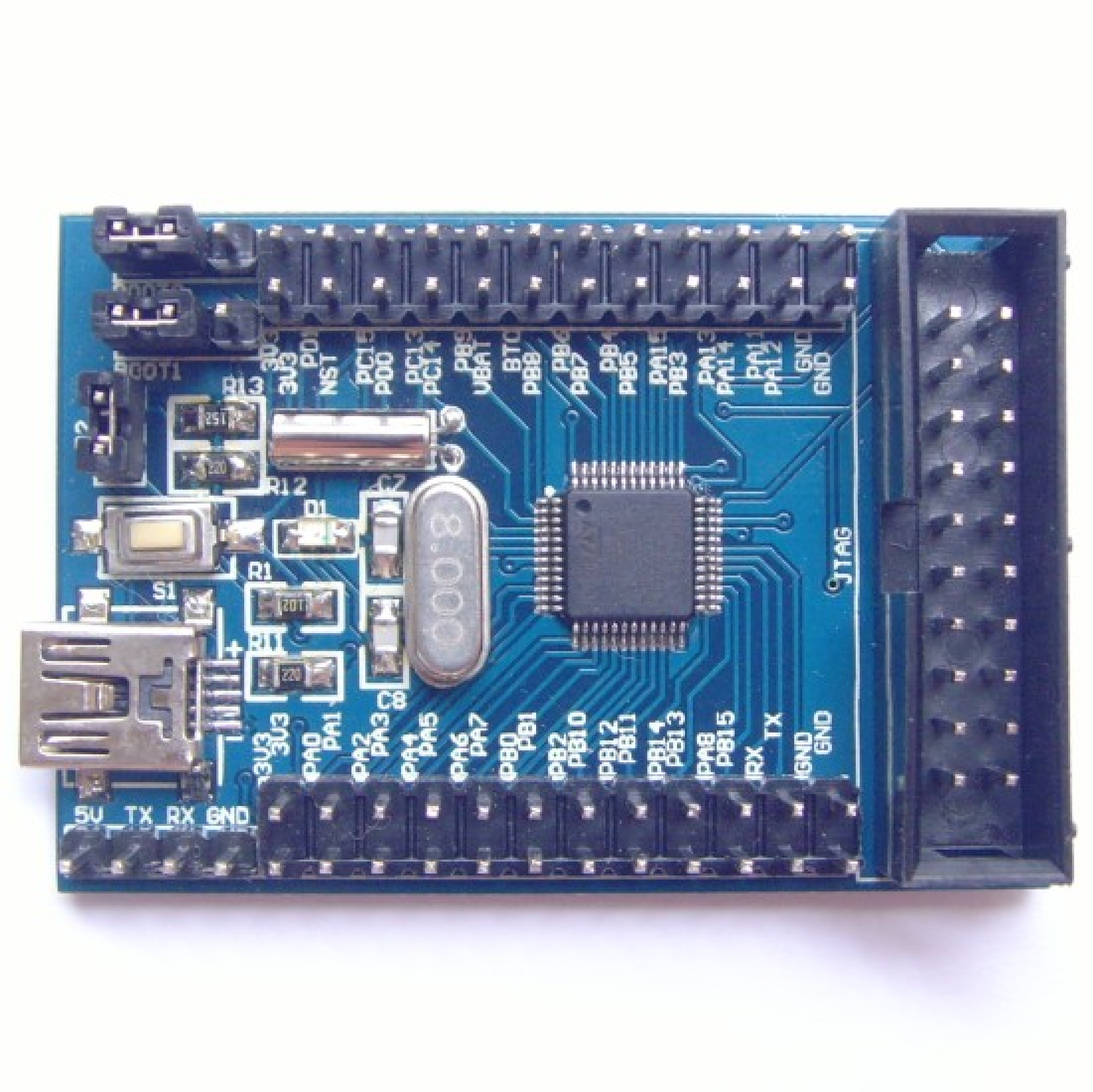
Warning

I see that a few of these boards now have rusty header pins, so very low quality solder coating. Are they any better than the Blue Pill which appeared later ? No, I don’t think so, probably very similar in quality.

Specs

- Price: Around $4 - $9
- Manufacturer: Shenzhen-LC
- MCU: STM32F103C8T6 Cortex-M3
- Resources: 64 Flash, USB, 8MHz crystal, *RTC crystal fitted*

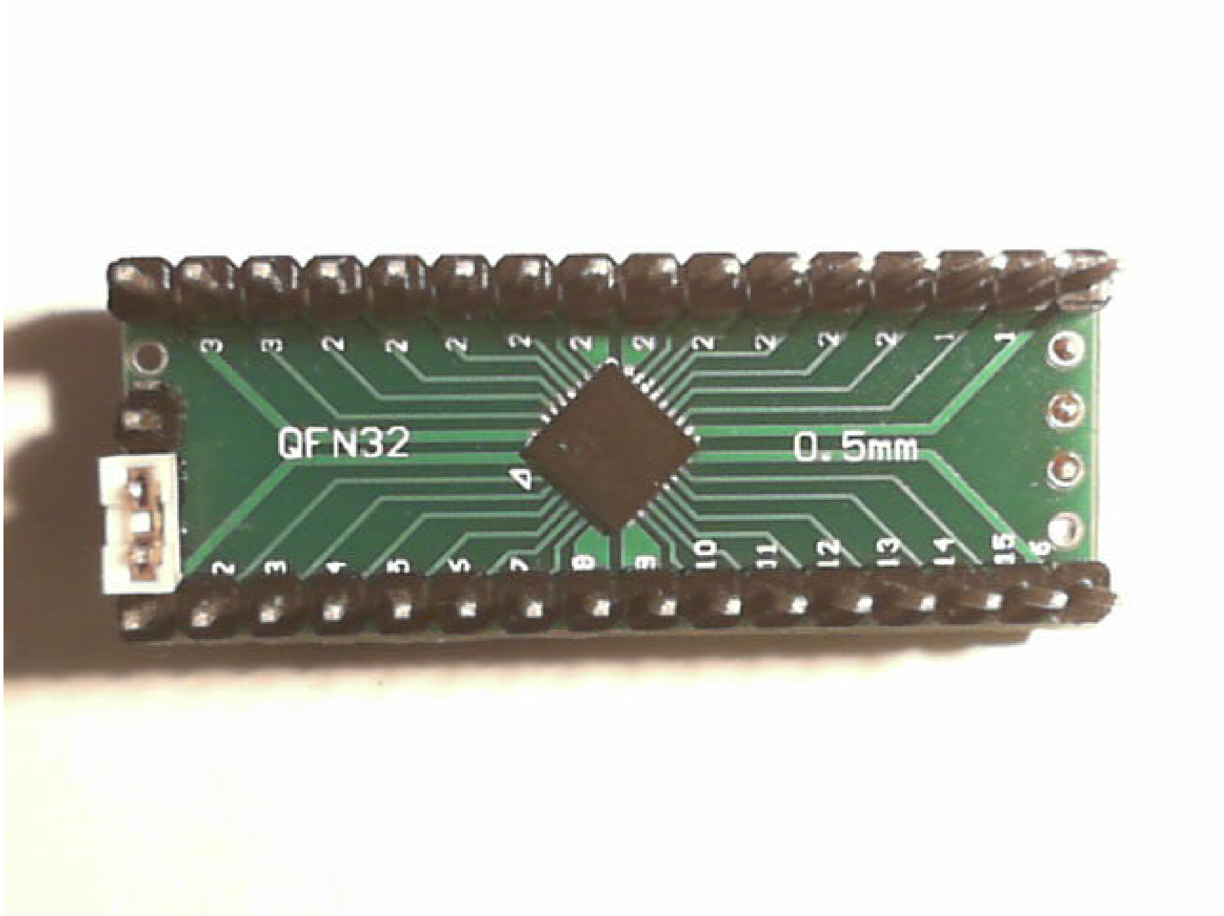
- Flashing: Serial Bootloader, and JTAG see Mecrisp-Stellaris serial connections and the Serial Bootloader Howto


BOOT1
R13
R12
D1
C7
S1
R1
R11
C8
JTAG
5V TX RX GND
3V3 3V3 PD1 NST PC15 PD0 PC13 PC14 PB9 VBAT BT0 PB8 PB6 PB7 PB4 PB5 PA15 PB3 PA13 PA14 PA11 PA12 GND GND
3V3 3V3 PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 PB0 PB1 PB2 PB10 PB12 PB11 PB14 PB13 PA8 PB15 RX TX GND GND

Why not make your own “GREEN PILL”

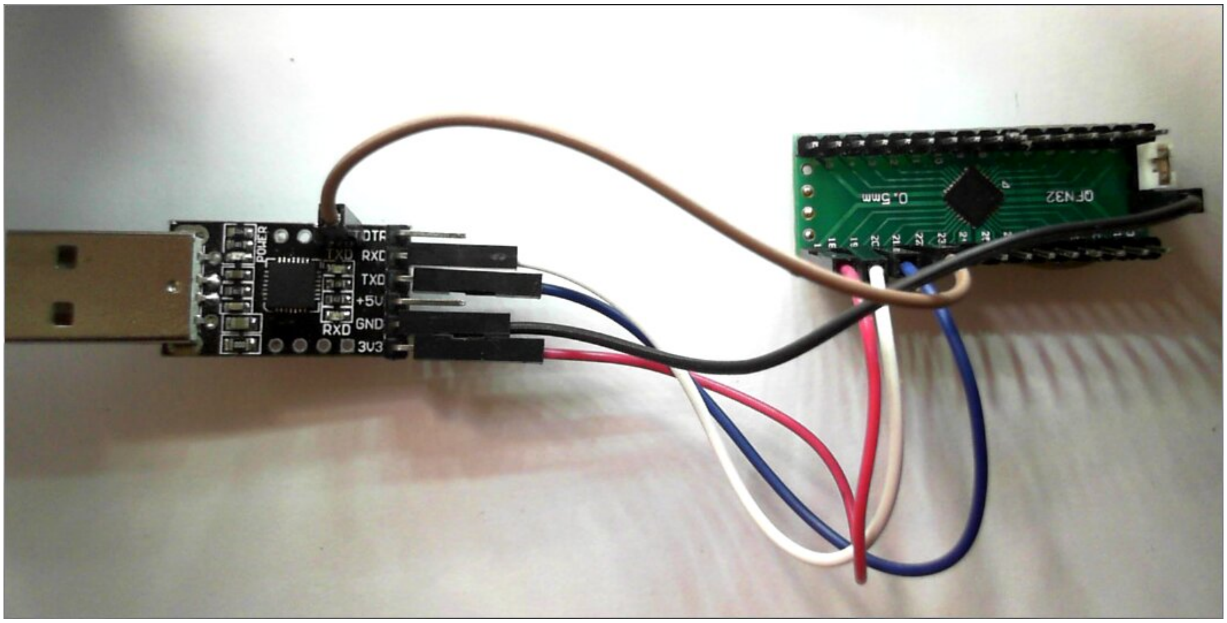
- Under $1.50 for **brand new authentic parts**
- Use a modern STM32F051K8 MCU, not a 12 year OLD STM32F103
- The board is powered by and talks serially to a PC at 460800 Baud via a $0.90 USB/3.3 Volt dongle

- Uses internal 8MHz RC clock.
- Only one external component on the board, a 47uF, 6v decoupling PSU capacitor.
- Mecrisp-Stellaris was flashed to the chip via the Serial Bootloader method after moving the WHITE option jumper from the “run” to the “bootload” position.



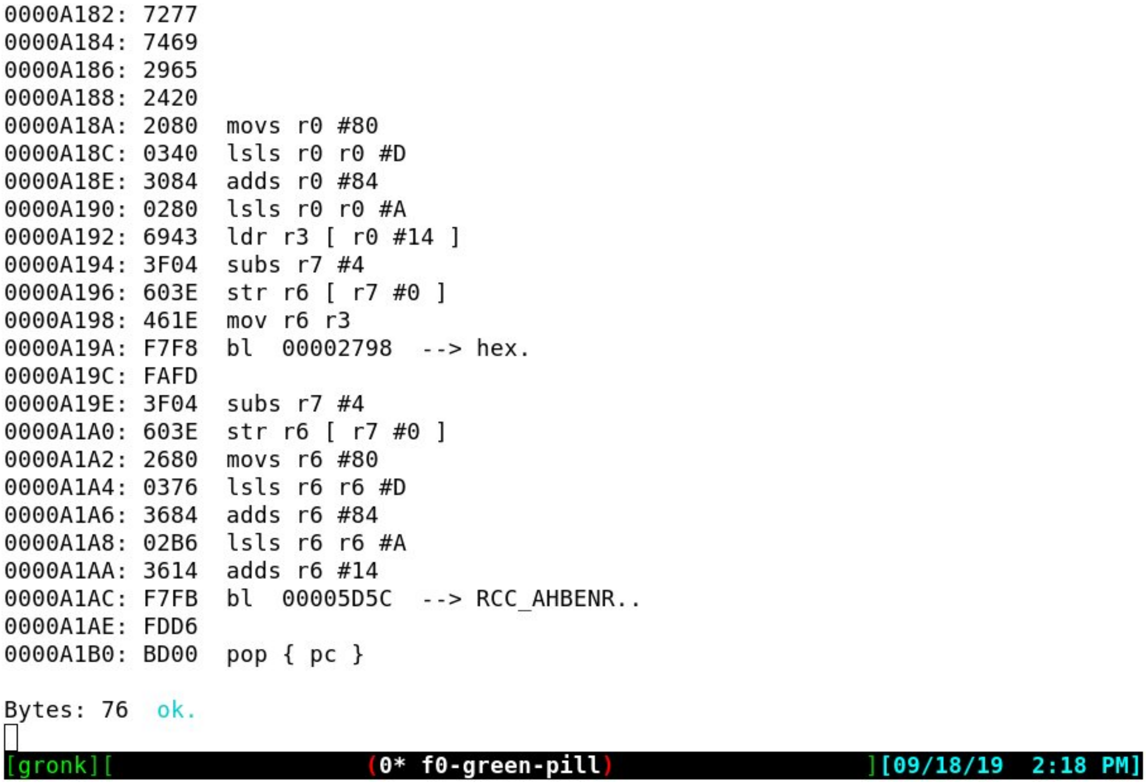
Green Pill Running Forth

This is the complete system.



The Terminal Screen

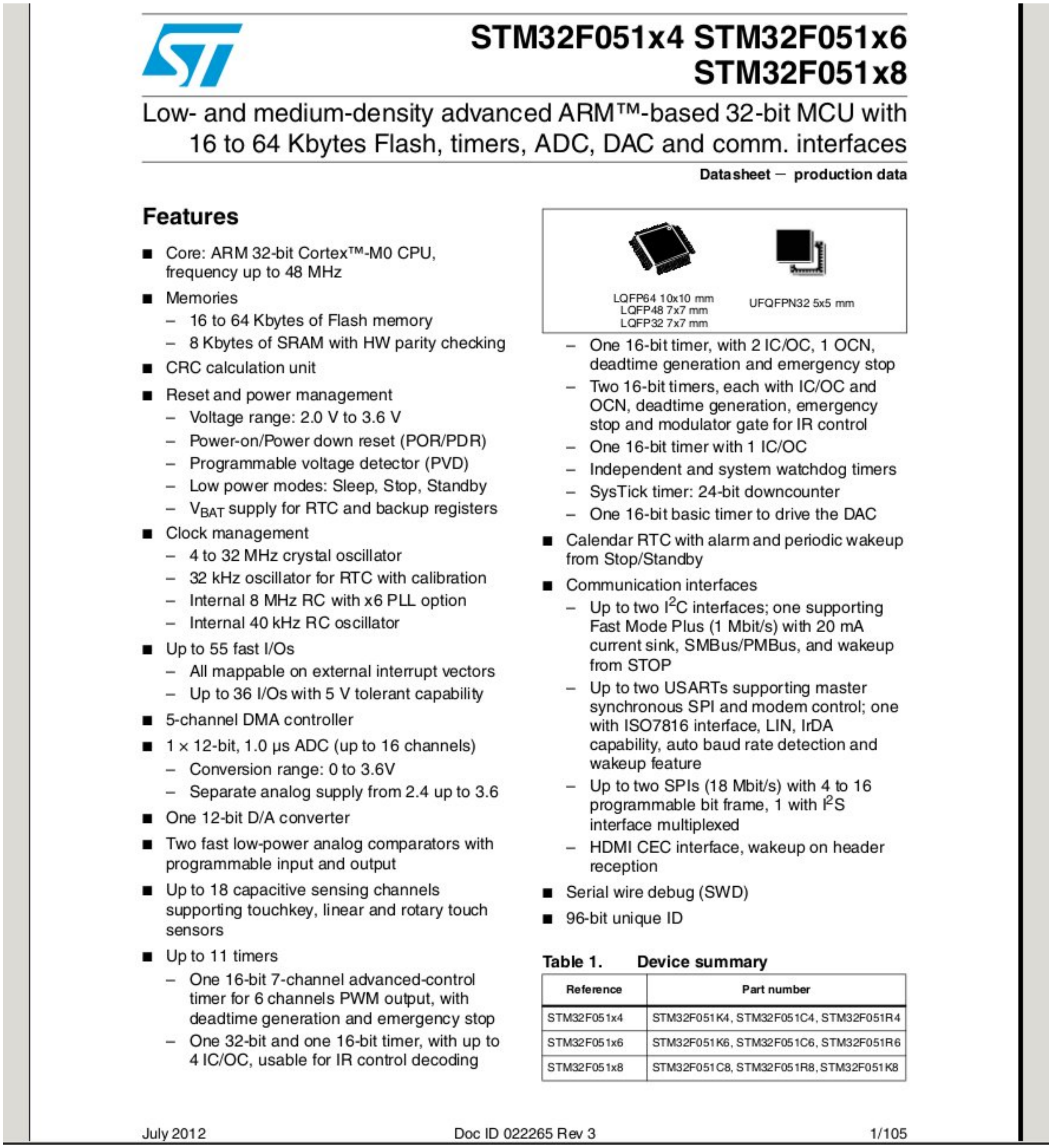
```
reset MS-RA_2.5.1_tpMOD with M0 core for STM32F051 by Matthias Koch
free (bytes)
FLASH.. TOTAL REPORTED: 65536 USED: 63488 FREE: 2048
RAM.... TOTAL PRESET: 8192 USED: 1180 FREE: 7012

 ok.
RCC_AHBENR.
RCC_AHBENR (read-write) $005E0000
                  I   I I I I                         F
              T   O   O O O O                         L   S
              S   P   P P P P                     C   I   R   D
              C   F   D C B A                     R   T   A   M
              E   E   E E E E                     C   F   M   A
              N   N   N N N N                     E   E   E   E
             |2| |2| |2|1|1|1|                    N   N   N   N
~|~|~|~|~|~|~|4|~|2|~|0|9|8|7|~|~|~|~|~|~|~|~|~|~|6|~|4|~|2|~|0
0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
see RCC_AHBENR.
0000A166: B500  push { lr }
0000A168: F7F8  bl  000025EE  --> cr
0000A16A: FA41
0000A16C: F7F8  bl  000026C0  -->  .' RCC_AHBENR (read-write) $'
0000A16E: FAA8
0000A170: 5219
0000A172: 4343
0000A174: 415F
0000A176: 4248
0000A178: 4E45
0000A17A: 2052
0000A17C: 7228
0000A17E: 6165
0000A180: 2D64
```



```
0000A182: 7277
0000A184: 7469
0000A186: 2965
0000A188: 2420
0000A18A: 2080  movs r0 #80
0000A18C: 0340  lsls r0 r0 #D
0000A18E: 3084  adds r0 #84
0000A190: 0280  lsls r0 r0 #A
0000A192: 6943  ldr r3 [ r0 #14 ]
0000A194: 3F04  subs r7 #4
0000A196: 603E  str r6 [ r7 #0 ]
0000A198: 461E  mov r6 r3
0000A19A: F7F8  bl  00002798  --> hex.
0000A19C: FAFD
0000A19E: 3F04  subs r7 #4
0000A1A0: 603E  str r6 [ r7 #0 ]
0000A1A2: 2680  movs r6 #80
0000A1A4: 0376  lsls r6 r6 #D
0000A1A6: 3684  adds r6 #84
0000A1A8: 02B6  lsls r6 r6 #A
0000A1AA: 3614  adds r6 #14
0000A1AC: F7FB  bl  00005D5C  --> RCC_AHBENR..
0000A1AE: FDD6
0000A1B0: BD00  pop { pc }

Bytes: 76  ok.

[gronk][                   (0* f0-green-pill)                   ][09/18/19  2:18 PM]
```

Chip

Features



STM32F051x4 STM32F051x6 STM32F051x8

Low- and medium-density advanced ARM™-based 32-bit MCU with 16 to 64 Kbytes Flash, timers, ADC, DAC and comm. interfaces

Datasheet — production data

Features

- Core: ARM 32-bit Cortex™-M0 CPU, frequency up to 48 MHz
- Memories
  - 16 to 64 Kbytes of Flash memory
  - 8 Kbytes of SRAM with HW parity checking
- CRC calculation unit
- Reset and power management
  - Voltage range: 2.0 V to 3.6 V
  - Power-on/Power down reset (POR/PDR)
  - Programmable voltage detector (PVD)
  - Low power modes: Sleep, Stop, Standby
  - V_{BAT} supply for RTC and backup registers
- Clock management
  - 4 to 32 MHz crystal oscillator
  - 32 kHz oscillator for RTC with calibration
  - Internal 8 MHz RC with x6 PLL option
  - Internal 40 kHz RC oscillator
- Up to 55 fast I/Os
  - All mappable on external interrupt vectors
  - Up to 36 I/Os with 5 V tolerant capability
- 5-channel DMA controller
- 1 × 12-bit, 1.0 µs ADC (up to 16 channels)
  - Conversion range: 0 to 3.6V
  - Separate analog supply from 2.4 up to 3.6
- One 12-bit D/A converter
- Two fast low-power analog comparators with programmable input and output
- Up to 18 capacitive sensing channels supporting touchkey, linear and rotary touch sensors
- Up to 11 timers
  - One 16-bit 7-channel advanced-control timer for 6 channels PWM output, with deadtime generation and emergency stop
  - One 32-bit and one 16-bit timer, with up to 4 IC/OC, usable for IR control decoding
  - One 16-bit timer, with 2 IC/OC, 1 OCN, deadtime generation and emergency stop
  - Two 16-bit timers, each with IC/OC and OCN, deadtime generation, emergency stop and modulator gate for IR control
  - One 16-bit timer with 1 IC/OC
  - Independent and system watchdog timers
  - SysTick timer: 24-bit downcounter
  - One 16-bit basic timer to drive the DAC
- Calendar RTC with alarm and periodic wakeup from Stop/Standby
- Communication interfaces
  - Up to two I^2C interfaces; one supporting Fast Mode Plus (1 Mbit/s) with 20 mA current sink, SMBus/PMBus, and wakeup from STOP
  - Up to two USARTs supporting master synchronous SPI and modem control; one with ISO7816 interface, LIN, IrDA capability, auto baud rate detection and wakeup feature
  - Up to two SPIs (18 Mbit/s) with 4 to 16 programmable bit frame, 1 with I^2S interface multiplexed
  - HDMI CEC interface, wakeup on header reception
- Serial wire debug (SWD)
- 96-bit unique ID

LQFP64 10x10 mm
LQFP48 7x7 mm
LQFP32 7x7 mm

UFQFPN32 5x5 mm

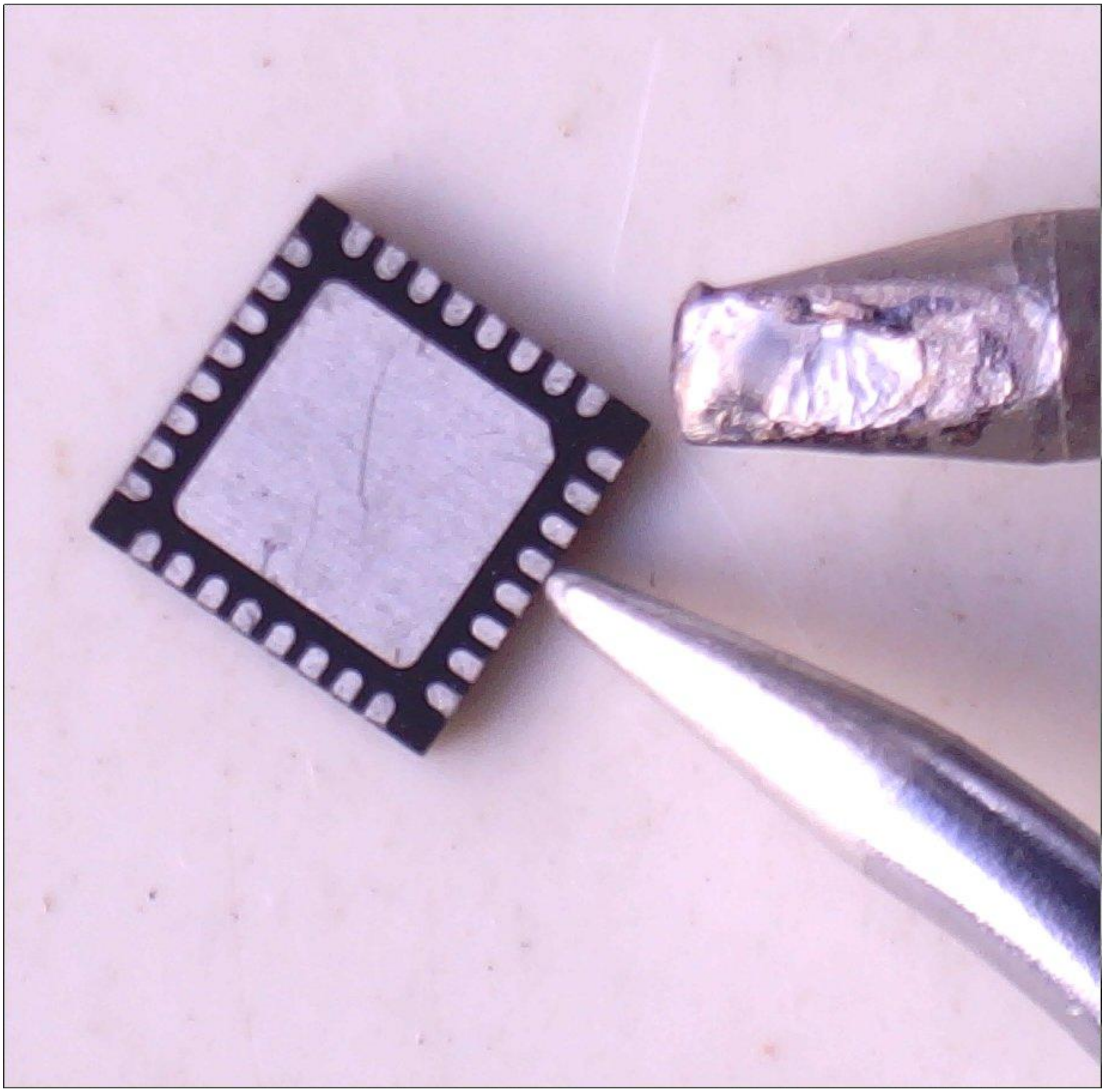
Table 1. Device summary

Reference	Part number
STM32F051x4	STM32F051K4, STM32F051C4, STM32F051R4
STM32F051x6	STM32F051K6, STM32F051C6, STM32F051R6
STM32F051x8	STM32F051C8, STM32F051R8, STM32F051K8

July 2012 Doc ID 022265 Rev 3 1/105

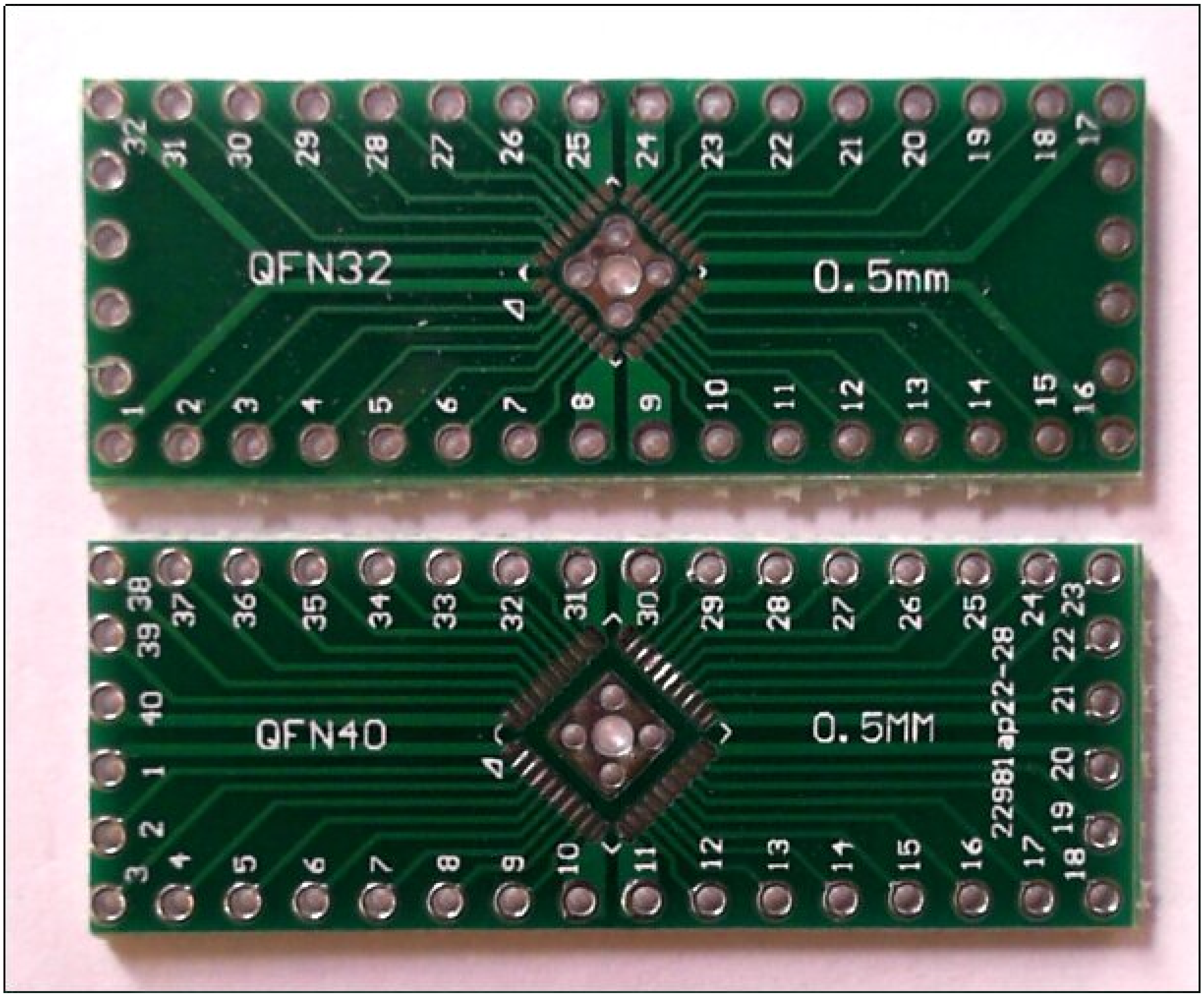
STM32F051K MCU (QFN32 Package)

Showing small soldering iron tip used to solder the MCU to the pcb.

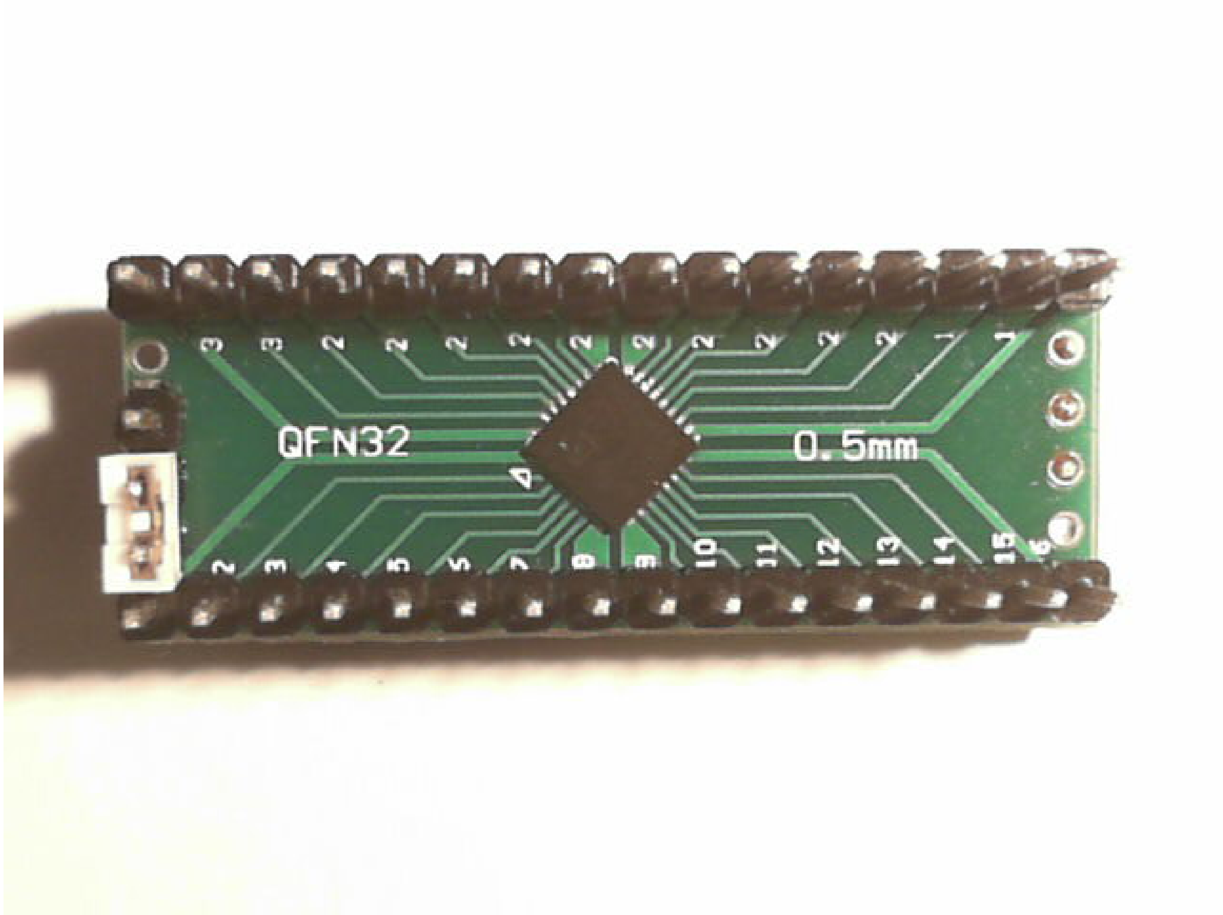


$0.20 Generic QFN40/32 Header Board

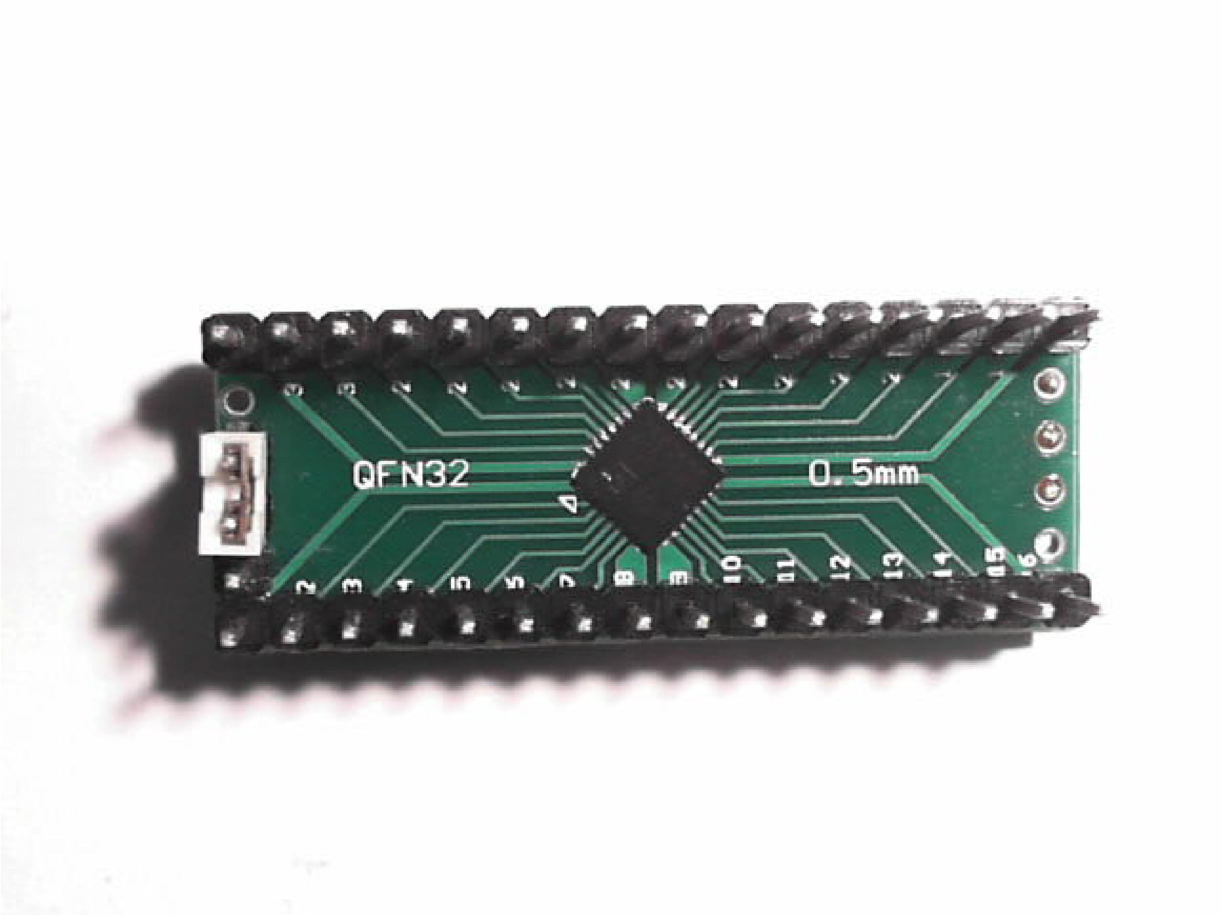
PCB Header: Top and bottom sides. The top is used for the MCU, the bottom for the wiring.


QFN32
0.5mm
QFN40
0.5MM
22981 ap22-28

Green Pill: Run Mode



Green Pill: Bootloader Mode

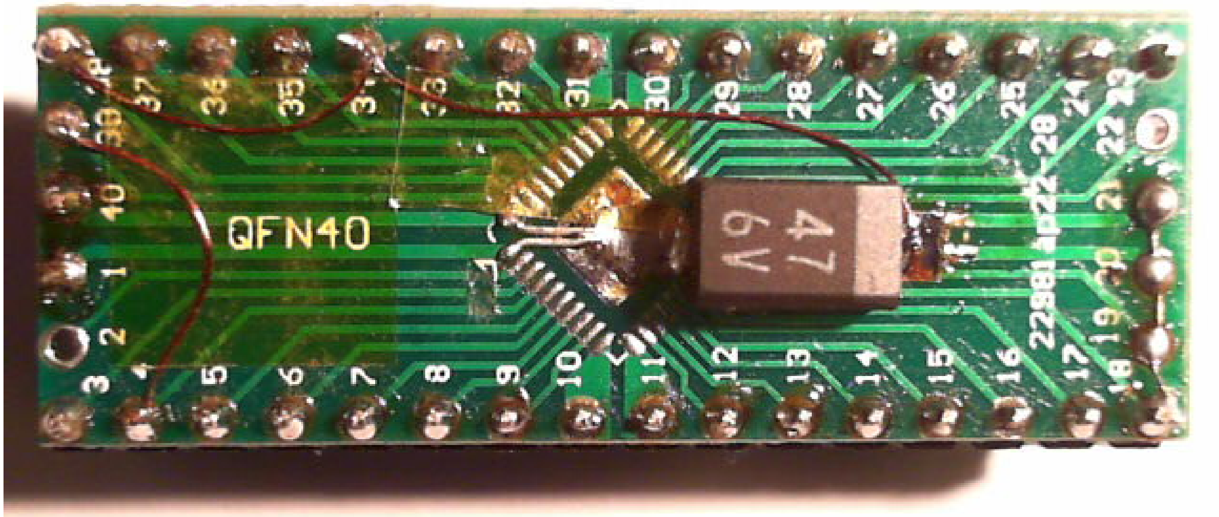


Green Pill Bottom View

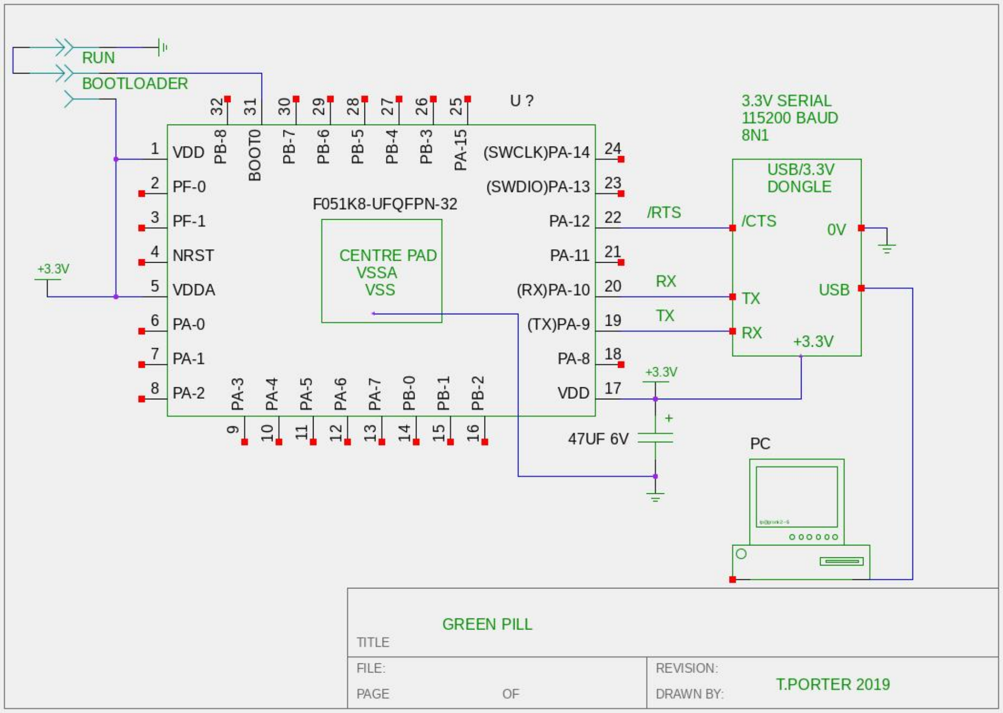
One 47uF 6V filter capacitor on the bottom.

Note

- **No cutting of tracks is needed**, just solder the wires as in the pic and according to the schematic. The picture is of my very first prototype, where various options were explored.
- Ignore the numbers on the bottom of the PCB, they belong to the QFN40 alternate layout, which isn't used here. The top QFN32 numbers correspond to the schematic.



Green Pill Schematic


RUN
BOOTLOADER
U ?
3.3V SERIAL
115200 BAUD
8N1
F051K8-UFQFPN-32
CENTRE PAD
VSSA
VSS
1 VDD
2 PF-0
3 PF-1
4 NRST
5 VDDA
6 PA-0
7 PA-1
8 PA-2
+3.3V
9 PA-3
10 PA-4
11 PA-5
12 PA-6
13 PA-7
14 PB-0
15 PB-1
16 PB-2
32 PB-8
31 BOOT0
30 PB-7
29 PB-6
28 PB-5
27 PB-4
26 PB-3
25 PA-15
24 (SWCLK)PA-14
23 (SWDIO)PA-13
22 PA-12
21 PA-11
20 (RX)PA-10
19 (TX)PA-9
18 PA-8
17 VDD
/RTS
RX
TX
USB/3.3V
DONGLE
/CTS
0V
TX
USB
RX
+3.3V
+3.3V
+
47UF 6V
PC
GREEN PILL
TITLE
FILE:
PAGE
OF
REVISION:
DRAWN BY:
T.PORTER 2019

STM32 Links

STM32 Wiki

https://en.wikipedia.org/wiki/STM32

STM32F Cortex M0 manual

http://www.st.com/resource/en/reference_manual/dm00031936.pdf

Cortex Microcontroller Software Interface Standard (CMSIS)

https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

GNU Binutils

GNU Assembler

https://sourceware.org/binutils/docs/as/

STM32F103C8 V/S GD32VF103Cx

These two MCU’s have totally different CORES, yet many of the PERIPHERALS are identical and both have Mecrisp Forth support, so this is a comparison page.

Mecrisp Support

MCU	Mecrisp Variant	Status	Tarball URL
STM32F103C8	Mecrisp-Stellaris	Stable	https://sourceforge.net/projects/mecrisp/files/mecri stellaris-2.5.2.tar.gz/download
GD32VF103Cx	Mecrisp-Quintus	Experimental	https://sourceforge.net/projects/mecrisp/files/mecri quintus-0.26-experimental.tar.gz/download

Note

Mecrisp-Quintus for the GD32VF103Cx is still experimental because the INTERRUPT support is not finalised.

Specifications

[quote author=SiliconWizard link=topic=225582.msg2877120#msg2877120 date=1579398691] [quote author=techman-001 link=topic=225582.msg2876628#msg2876628 date=1579371955] The Gigadevices RISC-V GD32VF103 (which also uses most of the F1 peripherals) is quite interesting, I’m playing with one at the moment. https://mecrisp-stellaris-folkdoc.sourceforge.io/gd32vf103.html#gd32vf103-longan-nano [/quote]

Just a small correction: you mention that the conditional branches range is +/-2048 bytes, which is not exact. Since instructions can only be aligned to 4 bytes, or 2 bytes for compressed instructions, the 12-bit offset actually encodes an offset that is in multiples of 2 bytes, which is clever, so the range is +/-4096 bytes. (I’m currently actively working on RISC-V so that piqued my attention!)

Otherwise, this little MCU seems nice. One thing I’m wondering about, and haven’t really seen figures, is the power consumption. If you have a board, could you measure the current draw while the MCU executes code? I’m just curious. [/quote]

Thanks for the conditional branch info, I’ll update my page :)

I’ll set something up to measure the current drain, all I can really do that be be vaguely informative is a comparison under similar circumstances with a STM32F103, both running a Forth primes program and post the results here and on my doc site.

Parameter	**STM32F103C8**	**GD32VF103Cx**
Architecture	32-bit ARM Cortex M3	32 bit RISC-V
Operating Voltage	2.7V to 3.6V	2.6v to 3.6V
CPU Frequency	72 MHz	108 MHz
Number of GPIO pins	37	37 ?
Number of PWM pins	12	?
Analog input Pins	10 (12-bit)	?
ADC	1	2
DAC	0	1
SDIO	1	0
USART	3	3
UART	0	2
I2C	2	2
SPI	2	3
Can 2.0	1	2
Timers	4	7
Flash Memory	64kB stated, usually 128kB	128kB
RAM	20kB	32kB
Peripherals	33	46

Parameter	STM32F103C8	GD32VF103Cx
Registers	466	1074
Bitfields	3503	7713

Peripherals

STM32F103C8	STM32F103C8 Peripheral	GD32VF103Cx	GD32VF103Cx Peripheral
.		ADC0	
ADC1		ADC1	
AFIO	Alternate function I/O	AFIO	
BKP	Backup registers	BKP	
CAN		CAN0	
.		CAN1	
CRC		CRC	
.		DAC	12 bit digital to analog converter
DBG	Debug support	DBG	
DMA1		DMA0	
DMA2		DMA1	
.		EXMC	External memory controller (EXMC)

STM32F103C8	STM32F103C8 Peripheral	GD32VF103Cx	GD32VF103Cx Peripheral
EXTI	External interrupt/event controller	EXTI	External interrupt/event controller
FLASH		FMC	Flash memory controller (FMC)
GPIOA		GPIOA	
GPIOB		GPIOB	
GPIOC		GPIOC	
GPIOD		GPIOD	
I2C1		I2C0	
I2C2		I2C1	
IWDG	Independent watchdog	FWDGT	Free watchdog timer (FWDGT)
NVIC	Nested Vectored Interrupt Controller	ECLIC	Enhancement Core.Local Interrupt Controller (ECLIC)
PWR	Power control	PMU	Power management unit (PMU
RCC	Reset and clock control	RCU	Reset and clock unit (RCU
RTC	Real time clock	RTC	
SDIO	Secure digital input/output interface	.	
.		SPI0	

STM32F103C8	**STM32F103C8 Peripheral**	**GD32VF103Cx**	**GD32VF103Cx Peripheral**
SPI1		SPI1	
SPI2		SPI2	
TIM1	Advanced timer	TIMER0	Advanced-timers
TIM2	General purpose timer	TIMER1	General-purpose-timers
TIM3	General purpose timer	TIMER2	General-purpose-timers
TIM4	General purpose timer	TIMER3	General-purpose-timers
.		TIMER4	General-purpose-timers
.		TIMER5	Basic-timers
.		TIMER6	Basic-timers
USART1	Universal synchronous asynchronous receiver transmitter	USART0	Universal synchronous asynchronous receiver transmitter
USART2		USART1	
USART3		USART2	
.		UART3	Universal asynchronous receiver transmitter
.		UART4	
USB	Universal serial bus full-speed device	USBFS_DEVICE	USB on the go full speed device
.		USBFS_GLOBAL	USB full speed global registers

STM32F103C8	STM32F103C8 Peripheral	GD32VF103Cx	GD32VF103Cx Peripheral
.		USBFS_HOST	USB on the go full speed host
.		USBFS_PWRCLK	USB on the go full speed power and clock gating control
WWDG	Window watchdog	WWDGT	Window watchdog timer

ISA Comparisons

Description	RISC-V	Thumb
Conditional branches	• 12 bit	signed 8-bit (-256 to +256 bytes)
Register Load-Store Immediate		A base register Rn plus a 5-bit immediate
Peripherals reset to default	?	SYSRESETREQ

- RISC-V Conditional branch workaround: invert the sense of the test and conditionally branch over an unconditional branch with more offset bits.

Premade Boards

See also

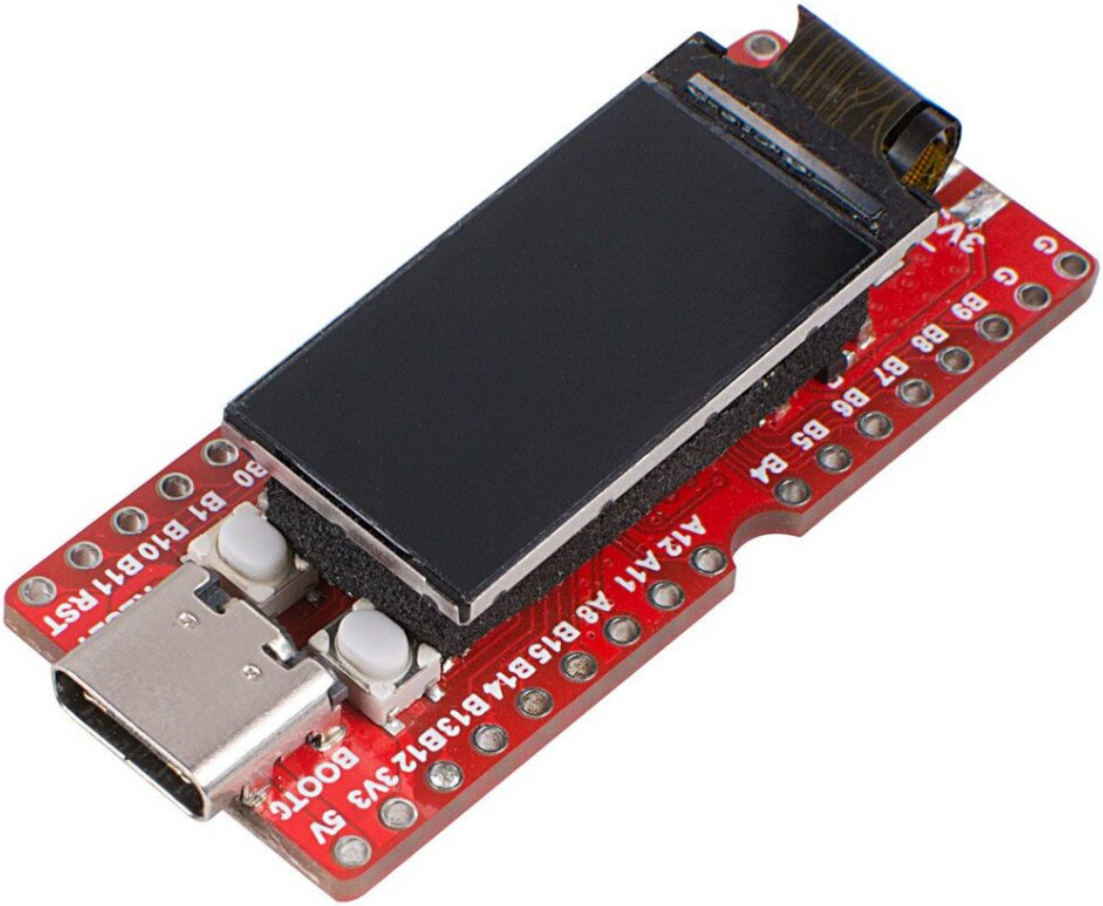
GD32VF103 RISC-V

RISC-V

Sipeed Longan Nano

RISC-V GD32VF103CBT6 Development Board

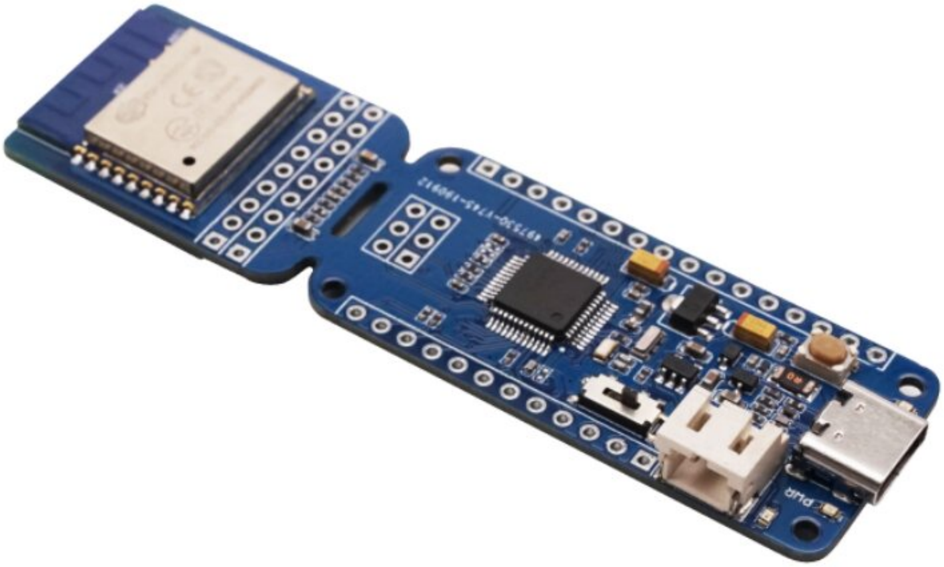
https://www.seeedstudio.com/Sipeed-Longan-Nano-RISC-V-GD32VF103CBT6-Development-Board-p-4205.html $4.90 USD


B0
B1
B10
B11
RST
BOOT0
3V3
5V
B12
B13
B14
B15
A8
A11
A12
B4
B5
B6
B7
B8
B9
G
G
3V

Wio Lite RISC-V (GD32VF103) - With ESP8266

Wio Lite RISC-V is a feather form factor RISC-V development board Based on GD32VF103, with the onboard ESP8266 Wio Core, it also features WiFi function.

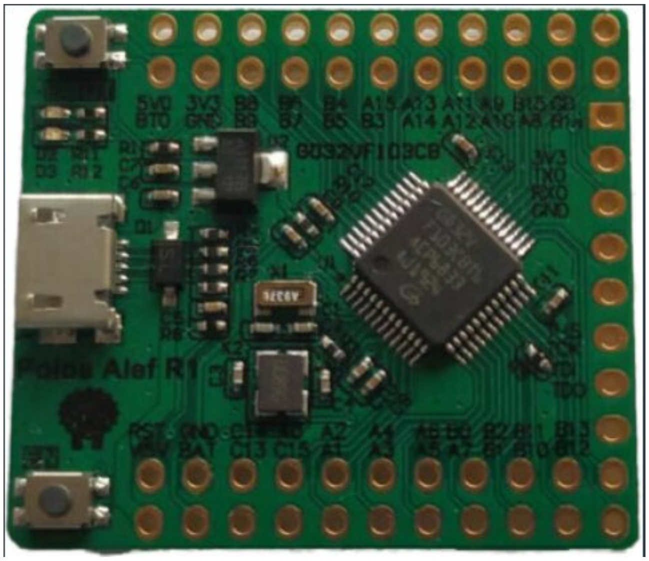
https://www.seeedstudio.com/Wio-Lite-RISC-V-GD32VF103-p-4293.html $6.90 USD



Polos GD32V Alef

Polos GD32VF103 Alef Board – RISC-V MCU Board with 128KB Flash 32KB RAM

https://www.analoglamb.com/product/polos-gd32v-alef-board-risc-v-mcu-board/ $2.99 USD



STM32F103C8

Blue Pill

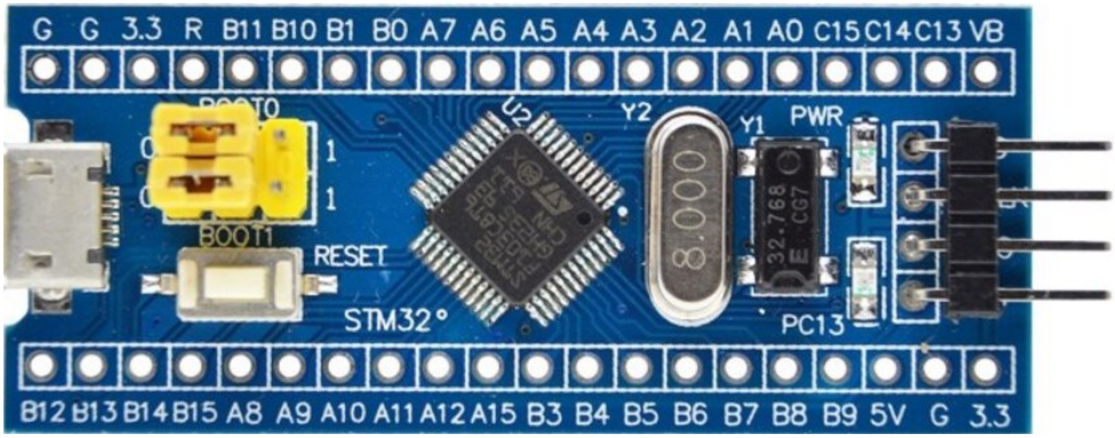
Available everywhere on the Internet around $2USD

See also

Blue Pill Rant

See also

Blue Pill Starter Kit


G
G
3.3
R
B11
B10
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
C15
C14
C13
VB
BOOT0
0
1
0
1
BOOT1
RESET
U2
STM32°
Y2
8.000
Y1
32.768
PWR
PC13
B12
B13
B14
B15
A8
A9
A10
A11
A12
A15
B3
B4
B5
B6
B7
B8
B9
5V
G
3.3

STM32F103C8T6, 2 x 64kB flash ?

Technically, only the **STM32F103CBT6** contains 128kB of Flash, the STM32F103C8T6 has only 64kB according to the STM device specs.

The very popular Bluepill Board contains a MCU marked “STM32F103C8T6”, with many units having 128kB of Flash. This has caused a belief that STM put 128kB of Flash in **every** STM32F103C8T6.

But is this true ?

YES, it sure looks like it is, read on …

STM32F103C8T6 #1 with 128kB Flash

This Bluepill has a MCU marked “STM32F103C8” and clearly runs a Mecrisp-Stellaris 90kB Forth binary proving it has 128kB of Flash. Data is from EEVblog. ChipID not known.

```
ramfree . 19112 ok.
flashfree . -24692 ok.
free  (bytes)
FLASH.. TOTAL REPORTED: 65536 USED: 90228 FREE: -24692
RAM.... TOTAL PRESET: 20000 USED: 888 FREE: 19112
```

Compare a Olimex P103 Board containing a STM32F103CBT6 MCU

Running a slightly bigger binary of 97kB. The MCU shows 128kB as expected and flashed a 90kB binary using standard OpenOCD configs, as expected.

```
stm32id
Die xy coords: 107741012
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00555285  |  U R .
Lot_num ascii encoded [55:24]: 0x87237267  |  . # r g

ok.
free (bytes)
FLASH.. TOTAL REPORTED: 131072 USED: 97028 FREE: 34044
RAM.... TOTAL PRESET: 20000 USED: 892 FREE: 19108
```

STM32F103C8T6 #2 with 64kB Indicated Flash

This board has a MCU marked “STM32F103C8” and clearly fails to flash the 90kB binary, (using the standard OpenOCD configs) indicating it has only 64kB Flash … but does it ?

I own 10 of these Chinese lc-mini boards. They are not Bluepills, but are much the same.

Standard OpenOCD Config

The standard OpenOCD flashing utility reads the chip reported Flash size of 64kB and accepts it (as it should).

90kB Binary Failed

```
% ./flash-eb13e308316527be6a8963098c7291d5.sh
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
 ** Programming Started **
auto erase enabled
Info : device id = 0x20036410
Info : flash size = 64kbytes
Warn : no flash bank found for address 8010000
wrote 65536 bytes from file
./eb13e308316527be6a8963098c7291d5.bin in 2.032206s (31.493
KiB/s)
```

```
** Programming Finished **
** Verify Started **
Error: checksum mismatch - attempting binary compare
diff 0 address 0x08010000. Was 0xff instead of 0x00
diff 1 address 0x08010001. Was 0xff instead of 0x06
...
diff 127 address 0x0801007f. Was 0xff instead of 0x06
More than 128 errors, the rest are not printed.
** Verify Failed **
shutdown command invoked
```

62.6 kB Binary PASSED

```
./flash-ae9236b921613c07b1967f21ff20ee97.sh
Open On-Chip Debugger 0.10.0
 ** Programming Started **
auto erase enabled
Info : device id = 0x20036410
Info : flash size = 64kbytes
wrote 63488 bytes from file
./ae9236b921613c07b1967f21ff20ee97.bin in 3.485837s (17.786
KiB/s)
** Programming Finished **
** Verify Started **
verified 62672 bytes in 0.919840s (66.537 KiB/s)
** Verified OK **
** Resetting Target **
Info : JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477
(mfg: 0x23b (ARM Ltd.), part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32f1x.bs tap/device found: 0x16410041
(mfg: 0x020 (STMicroelectronics), part: 0x6410, ver: 0x1)
shutdown command invoked
```

ChipID

```
stm32id
Die xy coords: 108003154
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00545267  |  T R g
Lot_num ascii encoded [55:24]: 0x87195634  |  . . V 4

free (bytes)
```

```
FLASH.. TOTAL REPORTED: 65536 USED: 62672 FREE: 2864
RAM.... TOTAL PRESET: 20000 USED: 880 FREE: 19120
```

Getting 128kB Flash!

The standard OpenOCD flashing utility reads the reported chip Flash size and uses it, BUT modifying the configuration to direct OpeoOCD to **IGNORE** the reported chip Flash size and use 128kB instead results in the **90kB binary image loading!**

stm32f1x.cfg Modification

```
# flash size will be probed
set _FLASHNAME $_CHIPNAME.flash
# flash bank $_FLASHNAME stm32f1x 0x08000000 0 0 0
$_TARGETNAME
flash bank $_FLASHNAME stm32f1x 0x08000000 0x20000 0 0
$_TARGETNAME
```

Results

```
** Programming Started **
auto erase enabled
Info : device id = 0x20036410
Info : ignoring flash probed value, using configured bank
size
Info : flash size = 128kbytes
wrote 91136 bytes from file
./eb13e308316527be6a8963098c7291d5.bin in 4.948380s (17.986
KiB/s)
** Programming Finished **
** Verify Started **
verified 90240 bytes in 1.311107s (67.214 KiB/s)
** Verified OK **
** Resetting Target **
Info : JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477
(mfg: 0x23b (ARM Ltd.), part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32f1x.bs tap/device found: 0x16410041
```

```
(mfg: 0x020 (STMicroelectronics), part: 0x6410, ver: 0x1)
shutdown command invoked
```

ChipID

Note this is STM32F103C8T6 #2 the same ID as the one above that initially failed the 90kB binary flash!

My Forth “free” program is obviously going to have to be changed to accommodate these findings :)

```
stm32id
Die xy coords: 108003154
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00545267  | T R g
Lot_num ascii encoded [55:24]: 0x87195634  | . . V 4

ok.
free (bytes)
FLASH.. TOTAL REPORTED: 65536 USED: 90228 FREE: -24692
RAM.... TOTAL PRESET: 20000 USED: 888 FREE: 19112
```

All 10 of these STM32F103C8T6 powered :ref:`Chinese lc-mini boards<lc-mini>` successfully flashed the 90kB image, they *all* have 129kB of Flash.

Thanks “Bingo”!

My thanks to “Bingo600” from EEVblog for convincing me to try this out :)

Storage Space

Using Unused Dictionary Space

You can use the free RAM dictionary space as temporary storage without the need for any allocation. You get its (changing) start address fresh for every usage with “compiletoram here”.

```
compiletoram here hex. 2000038C  ok.
```

> What’s New ? <

Supported Hardware

Alphabetically Listed by MCU

MCU Manufacturer / Board Name	MCU
Analog Devices	ADuCM4050
Atmel SAME70 X-plained	ATSAME70Q21
Cypress CY8CKIT-049 4200	CY8C4245AXI
Cypress	CY8C4247AZI
Cypress	CY8C4248BZI
Cypress	CY8C5868AXI
Cypress	CY8C5888LTI
Silicon Labs Happy Gecko Starter Kit	EFM32HG322F64

MCU Manufacturer / Board Name	MCU
Silicon Labs Giant Gecko Starter Kit	EFM32GG990F1024
Freescale Freedom FRDM-KL25Z board	KL25Z128VLK4
Freescale Freedom FRDM-KL46Z board	KL46Z256
TI Stellaris Launchpad	LM4F120H5QR
TI Stellaris Chinese QF-LM4F232	LM4F232H5QC
NXP	LPC1114FBD48
NXP dual in-line package (dip)	LPC1114FN28
NXP dual in-line package (dip)	LPC1115FBD48
NXP experimental -erasetoflash	LPC845M301
TI MSP432 Launchpad	MSP432P401R

MCU Manufacturer / Board Name	MCU
Freescale Teensy 3.1	MK20DX256VLH7
Freescale Freedom FRDM-K64F board	MK64FN1M0
BBC Micro Bit UK computer edu device	nRF51822
Nordicsemi nRFgo Starter Kit	nRF51822
Nordicsemi	nrf52832
Nuvoton Nutiny-SDK-NUC123	NUC123
STM Nucleo F207ZG	STM32F207ZG
Espruino Pico	STM32F401CD
STM Nucleo F401RE	STM32F401RE
STM Nucleo F411RE	STM32F411RET6

MCU Manufacturer / Board Name	MCU
STM TSSOP20 chip	STM32F030F4
STM F0 Nucleo	STM32F030R8
STM	STM32F042F6
STM F0 Discovery	STM32F051R8
STM	STM32F072RB
STM VL Discovery	STM32F100RB
Blue Pill	STM32F103C8T6
Shenzen Mini-Arm board	STM32F103C8T6
STM	STM32F303K8
STM Nucleo F303RE	STM32F303RE
STM F3 Discovery	STM32F303VCT6

MCU Manufacturer / Board Name	MCU
STM	STM32G031F8
STM	STM32G031J6
STM	STM32G071RB
STM F4 Discovery	STM32F407VGT6
Numworks Calculator	STM32F412VGT6
STM F429 Discovery	STM32F429ZIT6
STM F746 Discovery	STM32F746NG
STM	STM32L031K6
STM L053 Discovery	STM32L053C8T6
STM Nucleo L073RZ	STM32L073RZ
STM	STM32L082KZ

MCU Manufacturer / Board Name	**MCU**
STM Nucleo L152RE	STM32L152RE
STM Nucleo L432KC	STM32L432KC
STM L476 Discovery	STM32L476VG
STM STM32WB (with wireless) Nucleo	STM32WB55
Tiva Launchpad	TM4C123GH6PM
TI Tiva Connected Launchpad	TM4C1294NCPDT
Infineon XMC2GO board	XMC1100

FREEDOM OF ASSEMBLY

Svd2gas - The Freedom To Assemble

> Computer programming takes many forms, from the low level languages, to the HLL’s, from machine code to LISP and everywhere inbetween.

Machine Code

> The lowest level is called ‘Machine Code’ which are *hexadecimal numbers* and the only language that the CPU understands, so all programming languages eventually distill down to this, so that the CPU can understand what it has to do.
>
> While the CPU understands Machine Code perfectly, to us it looks like the streaming screens of glyphs, or *green rain* which featured so prominently in the film ‘The Matrix’.

Partial hexdump of mecrisp-stellaris-stm32f051.bin (Can you see the Woman in the Red Dress ?)

```
0000000 0330 2000 376d 0000 35fd 0000 35fd 0000
0000010 0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000020 0000 0000 0000 0000 0000 0000 3627 0000
0000030 0000 0000 0000 0000 3627 0000 35db 0000
0000040 3627 0000 3627 0000 3627 0000 3627 0000
0000050 3627 0000 364b 0000 366f 0000 3695 0000
0000060 3627 0000 3627 0000 3627 0000 3627 0000
0000070 36b7 0000 36db 0000 36ff 0000 3721 0000
0000080 3745 0000 3627 0000 3627 0000 3627 0000
0000090 3627 0000 3627 0000 3627 0000 3627 0000
```

Assembly Language

One step above Machine Code is a programming language called Assembly Language which is far more friendly to Humans. Here *hecadecimal numbers* are replaced by *Mnemonics* which are a tool to help us remember them, it’s similar to how a web url is easier to remember than a IP address, or used to be before machine generated URL’s became 500+ characters long.

Mecrisp-Stellaris Forth is written in Assembly code, here is a snippet from the STM32F051 Cortex M0 terminal.s file.

```
@ GPIO register map is similiar to STM32F4 chips.
.equ GPIOA_BASE      ,   0x48000000
.equ GPIOA_MODER     ,   GPIOA_BASE + 0x00
.equ GPIOA_AFRH      ,   GPIOA_BASE + 0x24

 @ Set PORTA pins 2 (TX) and 3 (RX) in AF1 for USART2
 ldr r1, = GPIOA_MODER
 ldr r0, = 0xA0
 str r0, [r1]

 @ Set alternate function 1 to enable USART1 pins 9 AND 10 on
Port A
 ldr r1, = GPIOA_AFRH
 ldr r0, = 0x110                   @ Alternate function 1 for
TX and RX pins of USART1 on PORTA
 str r0, [r1]
```

This is far more readable than Machine Code and far easier to write if you’re a Programmer. You don’t have to continually consult a opcode manual to find what the Hex number is for ‘ldr’ (load register) for example.

Assembly Language is far less *error prone* than Machine Code. The first commercial embedded project I made (1985), was a bottle filling controller, which filled 5000 bottles of hospital saline solution a day and consisted of about 1500 lines of machine code. If I recall correctly, that meant I also made about 1500 errors writing and debugging that code ;-)

A good Assembly Language programmer can generally write the fastest and smallest programs possible when compared to any other programming language.

Assembly Language programs are reasonably easy to *read* when commented well like the example above and this reads to a programmer like ‘go to the shop and buy 2 litres of chocolate ice-cream’ reads to non programmers.

This article isn’t actually about how to use the GNU Assembler because that alone would require vast amounts of your time, but a basic background helps to explain why I wrote svd2gas.

In the Mecrisp-Stellaris Forth Assembly Language snippet above, you can see the words ‘GPIOA_MODER’ and ‘GPIOA_AFRH’ which are STM32F MCU ‘Registers’, and these are just two of the hundreds, perhaps even thousands or Registers inside STM32F MCU’s depending on the model.

When the code says ‘ldr r1, = GPIOA_MODER’ it means ‘load r1 with the memory address of GPIOA_MODER.

We know that GPIOA_MODER lives at the memory location 0x48000000 from the ‘Equate Statement’ ‘.equ

GPIOA_MODER , GPIOA_BASE + 0x00’ and so on.

Having armed yourself with the GNU AS (assembler) manual, you’re now ready to start programming, or are you ? Is there a list of all the Equate Statements available, or will you have to write them all yourself by referring to the STM32F Cortex M0 manual ?

svd2gas

> Svd2gas will automatically generate files containing Assembly Language Equate and helpful Bitfield Statements using only the freely available CMSIS-SVD file for your STM32F Cortex M MCU. This provides a consistent naming strategy throughout your program, making code reuse and sharing that little bit easier.
>
> I wrote svd2gas after hacking the Mecrisp-Stellaris assembly source and noticing that there didn’t seem to be a freely available list of equate statements for Cortex M available on the Internet.

Note

Did you know that the complete equate statements file, (including bitfields) for a low end STM32F0xx Cortex M0 is 3987 lines long or that the same thing for a high end STM32F7x Cortex M4 is 17838 lines long ?

CMSIS

> Cortex Microcontroller Software Interface Standard (CMSIS)

The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.

CMSIS-SVD

System View Description for Peripherals.

Describes the peripherals of a device in an XML file and can be used to create peripheral awareness in debuggers or header files with peripheral register and interrupt definitions.

Assembler Toolchain

To assemble any code for Cortex-M you’ll require the arm-none-eabi-binutils package from your Linux or *bsd repository, Or you can install from a Tarball.

- arm-none-eabi-binutils GNU binutils for bare metal arm cross-development

svd2gas README

```
SVD2GAS, Copyright Terry Porter 2017 "terry@tjporter.com.au"
This program auto generates equate statements for all
registers in a STM32XX for arm-none-eabi-as using CMSIS-SVD
Licensed under the GPL, see http://www.gnu.org/licenses/
Developed on FreeBSD but also tested on a Linux Ubuntu
16.04.1 x64 VM

Howto
=====
* Untar this tarball to a directory
* Save your STM32Fxx.svd in it
* Edit the makefile line "FOLDED_SVD =" to suit your
STM32xx.svd
* Run make, and the following files will be created
a) STM32xx.svd.uf.svd, created from STM32xx.svd
b) template.xml, which is a list of all the registers in your
MCU.
c) STM32Fxx.svd.inc containing bit definitions for fields of
width 1, or shifts amounts for fields with a width greater
than 1 bit.

Comment out the registers you don't want in template.xml and
run make again to recreate the STM32Fxx.svd.inc file.
==============================================================
=====================================================

How to comment out registers in template.xml ?
==============================================
Not commented out:   <name
xmlns:xs="http://www.w3.org/2001/XMLSchema-
instance">RCC</name>
Commented out:  <!-- <name
xmlns:xs="http://www.w3.org/2001/XMLSchema-
instance">RCC</name> -->


Release contents
================
Makefile
mk.template.xsl
README.txt
svduf.xsl
svd2gas.xsl


Dependencies
============
make or gmake
xsltproc
vim (or any editor, change it in the Makefile)

Where to get STM32F SVD files ?
===============================
1) https://github.com/posborne/cmsis-svd
```

```
2) arm.com (free account required)
```

Getting svd2gas

`DOWNLOAD Svd2gas here`

Demo Assembly Blinky

This is a demo Blinky written in pure assembly language by the author of Mecrisp-Stellaris

Svd2forth-v3

Because Forth is interactive, the contents of all MCU Registers can be written or read in real time at the terminal. This feature is quite unique and the main reason that Forth is so productive during development.

Other systems such as C require GDB via SWD or JTAG, or a serial debugger working through a bootloader to enable the developer to see and change Register contents in real time.

However for a complex MCU such as the STM32F there are literally hundreds or thousands of Registers making easy reading of their contents a exercise in databook study and interpretation. Sure, a good designer has to know the databook intimately anyway, but fast and easy Peripheral/Register/Bitfield reading saves a lot of time and is what Svd2forth-v3 is all about.

Plus if you are new to STM32F, then Svd2forth-v3 may help reduce the inevitable information overload you will encounter.

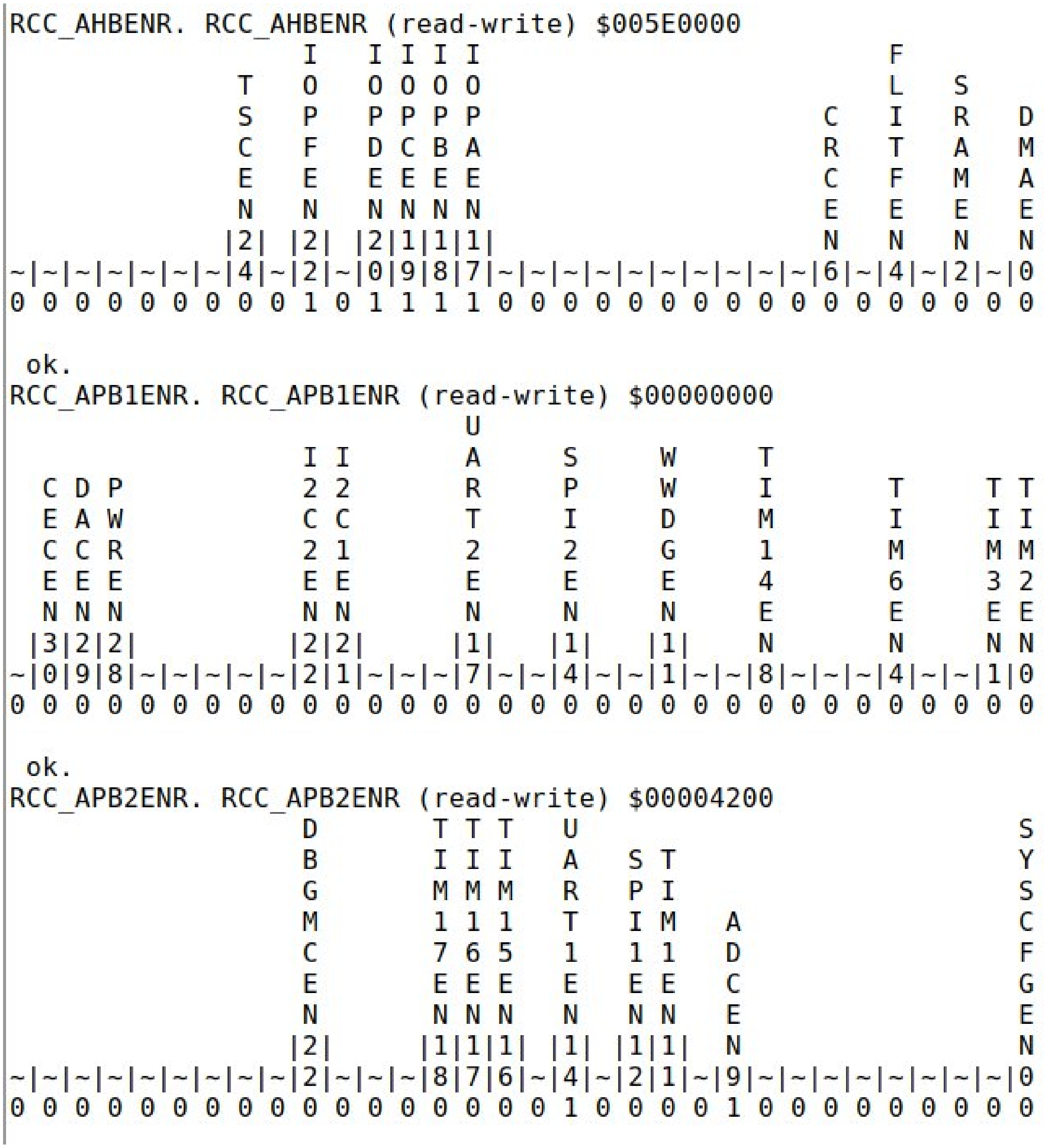
Svd2forth-v3 takes a CMSIS-SVD file created by STM and from it automatically creates memory-mapped Register, Register Bitfield and Pretty Printing Words.

Svd2forth-v3 Peripheral Register Pretty Printing

This screenpic shows a partial RCC Peripheral printout after entering “rcc.” The top third shows the RCC_ABHENR register configuration *right now* and it is obvious that IOP (GPIO) A,B,C,D,and F have been enabled by the “1” in the bottom line under each Bitfield.

Conversely, DMAEN is NOT enabled as it is configured to “0”.

The lowest third of this picture, RCC_APB2ENR shows that UART1 is enabled (UART1EN), as is the ADC (ADCEN) however they are the only ones.



```
RCC_AHBENR. RCC_AHBENR (read-write) $005E0000
                  I   I I I I                         F
              T   O   O O O O                         L   S
              S   P   P P P P                     C   I   R   D
              C   F   D C B A                     R   T   A   M
              E   E   E E E E                     C   F   M   A
              N   N   N N N N                     E   E   E   E
             |2| |2| |2|1|1|1|                    N   N   N   N
~|~|~|~|~|~|~|4|~|2|~|0|9|8|7|~|~|~|~|~|~|~|~|~|~|6|~|4|~|2|~|0
0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
RCC_APB1ENR. RCC_APB1ENR (read-write) $00000000
                            U
                  I I       A     S     W     T
  C D P           2 2       R     P     W     I       T     T T
  E A W           C C       T     I     D     M       I     I I
  C C R           2 1       2     2     G     1       M     M M
  E E E           E E       E     E     E     4       6     3 2
  N N N           N N       N     N     N     E       E     E E
 |3|2|2|         |2|2|     |1|   |1|   |1|    N       N     N N
~|0|9|8|~|~|~|~|~|2|1|~|~|~|7|~|~|4|~|~|1|~|~|8|~|~|~|4|~|~|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
RCC_APB2ENR. RCC_APB2ENR (read-write) $00004200
                  D       T T T   U                           S
                  B       I I I   A   S T                     Y
                  G       M M M   R   P I                     S
                  M       1 1 1   T   I M   A                 C
                  C       7 6 5   1   1 1   D                 F
                  E       E E E   E   E E   C                 G
                  N       N N N   N   N N   E                 E
                 |2|     |1|1|1| |1| |1|1|  N                 N
~|~|~|~|~|~|~|~|~|2|~|~|~|8|7|6|~|4|~|2|1|~|9|~|~|~|~|~|~|~|~|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
```

Svd2forth-v3 differs from V2 in the following ways

?	**Differences**
1	Register Legends can be hand made to enable fast interpretation of real time contents
2	Pretty Printing Legends are separated from the XML file and now contained in a “fxxx-legends.fs” file. This will allow different MCU Legend files to be created.
3	There is no “Register-Reference” file, it has been replaced with “bitfields.fs”
4	Both bitfields.fs and memmap.fs include the ‘description’ field from the SVD. Hopefully this will reduce the need to have a databook open while coding.

These notes

Are specifically for a STM32F051 but will work with any other STM32Fxx with the correct SVD file, however the Register ‘Pretty Printing’ Legends (f051-legends.fs) are for a STM32F051. This can be modified for your MCU, email me if you need more information.

Howto

- Untar svd2forth-v3 to a directory and in it:
- Edit the Makefile and replace the STM32F0xx-tp1.svd named in “FOLDED_SVD” with the SVD FILE for the MCU you are using.
- run “make everything”, this creates template.xml, bitfields.fs, and memmap.fs

- Edit “template.xml” and COMMENT OUT the Peripherals you are NOT using

Commented out ?	
No	<name xmlns:xs=”http://www.w3.org/2001/XMLSchema-instance”>GPIOE</name>
Yes	<!– <name xmlns:xs=”http://www.w3.org/2001/XMLSchema-instance”>GPIOE</name> –>

- run “make”
  This will create new bitfields.fs and memmap.fs based on your edited template.xml file.

- Upload f051-legends.fs then memmap.fs to your MCU. If the Flash or Ram is filled, then perhaps you didn’t bother editing template.xml to reduce the size ?
- bitfields.fs are for copy and pasting into your Forth program to configure Register Bitfields.

Printing a Peripheral

For every Peripheral enabled in template.xml there will be Pretty Print words generated in memmap.fs. Some will have handmade custom Legends but most will have a default format. Assume GPIOA is enabled in template.xml.

Enter “gpioa.” into your terminal, note the PERIOD after the Peripheral name, this is the “print Peripheral” command.

Note

Bitfields are grouped according to the STM Reference Manual, some are one bit groups, such as GPIOA_OTYPER, others are two bit groupings such as GPIOA_MODER, or four bit groupings as in GPIOA_AFRL. The GPIOx grouping have been hand edited otherwise automatic grouping of one bit groups applies.

```
gpioa.
```

```
gpioa. GPIOA_MODER (read-write) $02A80000
15| 14| 13| 12| 11| 10| 09| 08| 07| 06| 05| 04| 03| 02| 01| 00
00 00 00 10 10 10 10 00 00 00 00 00 00 00 00 00

GPIOA_OTYPER (read-write) $00000000
1| 1| 1| 1| 1| 1|
5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

GPIOA_OSPEEDR (read-write) $0C000000
15| 14| 13| 12| 11| 10| 09| 08| 07| 06| 05| 04| 03| 02| 01| 00
00 00 11 00 00 00 00 00 00 00 00 00 00 00 00 00

GPIOA_PUPDR (read-write) $24000000
15| 14| 13| 12| 11| 10| 09| 08| 07| 06| 05| 04| 03| 02| 01| 00
00 10 01 00 00 00 00 00 00 00 00 00 00 00 00 00

GPIOA_IDR (read-only) $0000A5FA
1| 1| 1| 1| 1| 1|
5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0
1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 0

GPIOA_ODR (read-write) $00000000
1| 1| 1| 1| 1| 1|
5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

GPIOA_BSRR (write-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

GPIOA_LCKR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
GPIOA_AFRL (read-write) $00000000
 07   06   05   04   03   02   01   00
0000 0000 0000 0000 0000 0000 0000 0000

GPIOA_AFRH (read-write) $00011110
 15   14   13   12   11   10   09   08
0000 0000 0000 0001 0001 0001 0001 0000

GPIOA_BRR (write-only) $00000000
1|1|1|1|1|1|
5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

ok.
```

Printing a Register

Registers can be printed individually. Enter “GPIOA_IDR.” into your terminal, note the PERIOD after the Register name, this is the “print Register” command.

```
GPIOA_IDR.
```

```
GPIOA_ODR. GPIOA_ODR (read-write) $00000000
1|1|1|1|1|1|
5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

ok.
```

Using bitfields.fs in your code

There are two types of Bitfields generated:

1. ONE configuration bit only, i.e.

```
: GPIOF_OTYPER_OT15   %1 15 lshift GPIOF_OTYPER bi? ;  \
GPIOF_OTYPER_OT15    Port x configuration bit  15
```

This type of bitfield will usually SET or CLEAR the one bit with a “bis!” or “bic!” command, or it may possibly be tested with a “bit@” command. See the Dictionary for more information.

2. More than one configuration bit, i.e.

```
: GPIOF_MODER_MODER15   ( %XX -- ) 30 lshift GPIOF_MODER bis! ;
\ GPIOF_MODER_MODER15    Port x configuration bits y =  0..15
```

I’ve selected code readability over size, so I usually use multiple bit Bitfields (in combination with the above command) this way:

```
: GPIOF_MODER_MODER15   ( %XX -- ) 30 lshift GPIOF_MODER bis! ;
\ GPIOF_MODER_MODER15    Port x configuration bits y =  0..15
%01 GPIOF_MODER_MODER15
\ set GPIOF-15 as a OUTPUT
```

RCC Memmap.fs Example

This shows the contents of the RCC Peripheral section in memmap.fs. This is the file where Peripherals and Registers are memory mapped to the hardware, a bit like a C header file. It is also the file where Pretty Printing Words are created. This file is referenced by the Words in bitfields.fs and it references the Words in fxxx-legends.fs.

```
$40021000 constant RCC ( Reset and clock control )
RCC $0 + constant RCC_CR ( Clock control register )
RCC $4 + constant RCC_CFGR ( Clock configuration register
RCC_CFGR )
RCC $8 + constant RCC_CIR ( Clock interrupt register  RCC_CIR )
RCC $C + constant RCC_APB2RSTR ( APB2 peripheral reset register
RCC_APB2RSTR )
RCC $10 + constant RCC_APB1RSTR ( APB1 peripheral reset register
RCC_APB1RSTR )
RCC $14 + constant RCC_AHBENR ( AHB Peripheral Clock enable
register  RCC_AHBENR )
RCC $18 + constant RCC_APB2ENR ( APB2 peripheral clock enable
register  RCC_APB2ENR )
RCC $1C + constant RCC_APB1ENR ( APB1 peripheral clock enable
register  RCC_APB1ENR )
RCC $20 + constant RCC_BDCR ( Backup domain control register
RCC_BDCR )
RCC $24 + constant RCC_CSR ( Control/status register  RCC_CSR )
RCC $28 + constant RCC_AHBRSTR ( AHB peripheral reset register )
RCC $2C + constant RCC_CFGR2 ( Clock configuration register 2 )
RCC $30 + constant RCC_CFGR3 ( Clock configuration register 3 )
RCC $34 + constant RCC_CR2 ( Clock control register 2 )
```

```
: RCC_CR. ." RCC_CR () $" RCC_CR @ hex. RCC_CR RCC_CR.. ;
: RCC_CFGR. ." RCC_CFGR () $" RCC_CFGR @ hex. RCC_CFGR RCC_CFGR..
;
: RCC_CIR. ." RCC_CIR () $" RCC_CIR @ hex. RCC_CIR 1b. ;
: RCC_APB2RSTR. ." RCC_APB2RSTR (read-write) $" RCC_APB2RSTR @
hex. RCC_APB2RSTR 1b. ;
: RCC_APB1RSTR. ." RCC_APB1RSTR (read-write) $" RCC_APB1RSTR @
hex. RCC_APB1RSTR 1b. ;
: RCC_AHBENR. ." RCC_AHBENR (read-write) $" RCC_AHBENR @ hex.
RCC_AHBENR RCC_AHBENR.. ;
: RCC_APB2ENR. ." RCC_APB2ENR (read-write) $" RCC_APB2ENR @ hex.
RCC_APB2ENR RCC_APB2ENR.. ;
: RCC_APB1ENR. ." RCC_APB1ENR (read-write) $" RCC_APB1ENR @ hex.
RCC_APB1ENR RCC_APB1ENR.. ;
: RCC_BDCR. ." RCC_BDCR () $" RCC_BDCR @ hex. RCC_BDCR 1b. ;
: RCC_CSR. ." RCC_CSR () $" RCC_CSR @ hex. RCC_CSR RCC_CSR.. ;
: RCC_AHBRSTR. ." RCC_AHBRSTR (read-write) $" RCC_AHBRSTR @ hex.
RCC_AHBRSTR 15b. ;
: RCC_CFGR2. ." RCC_CFGR2 (read-write) $" RCC_CFGR2 @ hex.
RCC_CFGR2 RCC_CFGR2.. ;
: RCC_CFGR3. ." RCC_CFGR3 (read-write) $" RCC_CFGR3 @ hex.
RCC_CFGR3 RCC_CFGR3.. ;
: RCC_CR2. ." RCC_CR2 () $" RCC_CR2 @ hex. RCC_CR2 RCC_CR2.. ;
: RCC.
RCC_CR.
RCC_CFGR.
RCC_CIR.
RCC_APB2RSTR.
RCC_APB1RSTR.
RCC_AHBENR.
RCC_APB2ENR.
RCC_APB1ENR.
RCC_BDCR.
RCC_CSR.
RCC_AHBRSTR.
RCC_CFGR2.
RCC_CFGR3.
RCC_CR2.
;
```

RCC_CR2 bitfields.fs Example

Note

Replace ‘bi?’ with ‘bit@’ to test bit, ‘bis!’ to SET bit, ‘bic!’ to CLEAR bit etc.

```
\ RCC_CR2 ()
: RCC_CR2_HSI14ON    %1 0 lshift RCC_CR2 bi? ;              \
RCC_CR2_HSI14ON    HSI14 clock enable
: RCC_CR2_HSI14RDY   %1 1 lshift RCC_CR2 bi? ;              \
RCC_CR2_HSI14RDY    HR14 clock ready flag
: RCC_CR2_HSI14DIS   %1 2 lshift RCC_CR2 bi? ;              \
RCC_CR2_HSI14DIS    HSI14 clock request from ADC  disable
: RCC_CR2_HSI14TRIM   ( %XXXXX -- ) 3 lshift RCC_CR2 bi? ;     \
RCC_CR2_HSI14TRIM    HSI14 clock trimming
: RCC_CR2_HSI14CAL    ( %XXXXXXXX -- ) 8 lshift RCC_CR2 bi? ;  \
RCC_CR2_HSI14CAL    HSI14 clock calibration
```

RCC_CR2 Pretty Print Example

This example shows hand edited Legends using the syntax of the STM Reference Manual.

```
rcc_cr2.
```

```
RCC_CR2 () $00006780

                                                          H H
                                                          S S H
                                                          I I S
                                                          1 1 I
                                                          4 4 1
                               |HSICAL         |HSI14    |D R 4
                               |7:6:5:4:3:2:1:0|TRIM     |I|D|O
                               |1|1|1|1|1|1|   |4:3:2:1:0|S|Y|N
~|~|~|~|~|~|~|~|~|~|~|~|~|~|~|~|5|4|3|2|1|0|9|8|7|6|5|4|3|2|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0
```

Svd2forth Downloads

Svd2forth Downloads

> What’s New ? <

Serial Terminal: Hardware

Mecrisp Stellaris is programmed via a serial terminal, often running on a PC. Default communication parameters are serial, at 115200 baud, 8N1 and use the on MCU USART.

Serial Terminal Hardware requires Serial Terminal Software to work.

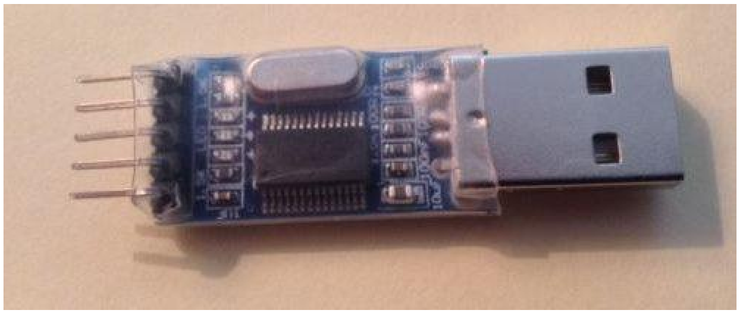
USB-3.3v Dongle or cables

> USB serial Dongles or cables are usually the easiest way to connect a PC running Picocom to a STM32 Discovery Board. Don’t forget to buy at least four female to female pin connecting cables for a dongle. These are usually available online in bundles.

Unix friendly CP2102 chipset dongle, with Rx, Tx, nRTS,nCTS and other control signals, including +5 and +3.3 volts available on pins or holes for pins or wires to be soldered. Usually under $2 on eBay, and the most useful model in my opinion

RST
R1
SUS
SUS
3V3
GND
+5V
TXD
RXD
DTR
CTS
RTS
DSR
DCD

Unix friendly PL2303 chipset dongle, with Rx, Tx, 3.3v, 5v available on pins at the rear, but no control signals, unless you want to hand solder fine wires to the SMT chip. Usually about $0.70 AUD each in a pack of 5 on Ebay



USB-3.3v cable. The cable length and flying leads make this a handy unit. About $25 on eBay

signal	**color**
+5v	Red
Ground	Black
TX	Orange (output)
RX	Yellow (input)
nCTS	Brown (input)
nRTS	Green (output)

Warning

Don’t use RS-232 cables or you may blow up your Board/MCU from overvoltage

Serial Pin Connection Table For Various Boards

Mecrisp-Stellaris default serial pin connections. These can be usually be changed in the Mecrisp-Stellaris assembly source code without too much effort.

Board	MCU	Type	TX	RX	MCU USB Available ?	Notes
Stellaris Launchpad	LM4F120	UART 0	?	?	Y	
Chinese QF-LM4F232	LM4F232H5QC	?	?	?	?	
Tiva Connected Launchpad	TM4C1294	USB	na	na	Y	Use connector U22
TI MSP432 Launchpad	MSP432P401R	USCI 0	P1.3	P1.2	Y	
LPC1114FBD48	LPC1114FBD48	UART	P1.7	P1.6		
LPC1114FN28	LPC1114FN28	UART	P1.7	P1.6	N	
LPC1115FBD48	LPC1115FBD48	UART	P1.7	P1.6	N	
STM32F030F4	STM32F030F4	USART1	PA9	PA10	N	
STM32F030K6	STM32F030K6	USART1	PA9	PA10	N	
STM32F0 Discovery	STM32F051R8	USART1	PA9	PA10	N	
STM L053 Discovery	STM32L053C8	USART1	PA9	PA10	N	

Board	MCU	Type	TX	RX	MCU USB Available ?	Notes
STM VL Discovery	STM32F100RB	USART1	PA9	PA10	N	
Blue Pill	STM32F103C8T6	USART1	PA9	PA10	Y	
Shenzhen LC mini STM32	STM32F103C8T6	USART1	PA9	PA10	Y	
STM Nucleo L152RE	STM32L152RE	USART2	PA9	PA10	Y	
STM F3 Discovery	STM32F303VCT6	USART1	PA9	PA10	N	
STM Nucleo 401RE	STM32F401RE	USART2	PA2	PA3	Y	
STM F429 Discovery	STM32F429ZIT6	USART1	PA9	PA10	N	
STM F4 Discovery	STM32F407VGT6	USART2	PA2	PA3	N	
STM Nucleo 411RE	STM32F411RET6	USART2	PA2	PA3	Y	
STM L476 Discovery	STM32L476VG	USART2	PD5	PD6	Y	
STM F746 Discovery	STM32F746NG	?	?	?	?	

Board	MCU	Type	TX	RX	MCU USB Available ?	Notes
Freescale Freedom FRDM?KL25Z	KL25Z128VLK4	UART0	?	?	Y	
KL46Z256	KL46Z256	UART0	?	?	Y	
Teensy 3.1	MK20DX256VLH7	UART0	Pin3	Pin2	?	Pin18/19 CTS/RTS
nRFgo Starter Kit	nRF51822	?	?	?	?	
nRF52832	nRF52832	UARTE	P02	P03	?	
NUC123	NUC123	UART0	?	?	?	64pin and 48pin only
Infineon XMC2GO	XMC1100Q024F0064	?	?	?	?	
EFM32 Giant Gecko	EFM32GG990F1024	?	?	?	?	

Note

See your mecrisp-stellaris-x.x.x/README for how to Flash your chip.

Note: Direct MCU USB access

Some MCU’s offer on chip USB peripherals for which Mecrisp-Sterllaris has limited support (see the list above).

Compared to using a direct MCU USB connection, using the MCU USART and a USB/3.3volt dongle may initially seem redundant but Forth project development involves regular resetting/rebooting of the MCU (as one screws up the hardware configs).

Warning

Resetting the MCU kills the USB peripheral which causes the USB device to be lost at the PC level with the subsequent death of the PC terminal application.

Note

With a USB/3.3volt dongle, you can reset the MCU as much as you like and your terminal application will stay blissfully ignorant and happy. You still have the convenience of USB with a USB/3.3volt dongle.

Serial Terminal: Hardware Flow Control

> This is now working, see my RTS patch on how to implement RTS Handshaking on a F0 Disco.

RTS/CTS flow control explained

- RTS means “Request To Send”. When this pin stays asserted (positive voltage) at the receiver it means: keep sending data to me.
- If RTS is negated (voltage goes negative) it negates “Request To Send” which means: request not to send to me (stop sending).
- When the receiver is ready for more input, it asserts RTS requesting the other side to resume sending.
- For computers and terminals (both DTE type equipment) the RTS pin sends the flow control signal to the CTS pin (Clear To Send) on the other end of the cable. That is, the RTS pin on one end of the cable is connected to the CTS pin at the other end.

Mecrisp-Stellaris Hardware used

- STM32F0 Discovery (Disco) Board

USB/3.3v Serial hardware Option 1

Unix friendly CP2102 chipset dongle, with Rx, Tx, nRTS,nCTS and other control signals, including +5 and +3.3 volts available on pins or holes for pins or

wires to be soldered. Usually under $2 on eBay, and the most useful model in my opinion

RI
3V3
GND
+5V
TXD
RXD
DTR
CTS
RTS
DSR
DCD

USB/3.3v Serial hardware Option 2

- $25 FTDI usb serial cable with RX, TX, GND, 5V DC supply and nCTS and nRTS signals on a 6 pin connector. FT232RQ chip is used. USB Vendor ID (VID): 0403h, USB Product UD (PID): 6001h . Note TX and RX have a internal 270R resistor.

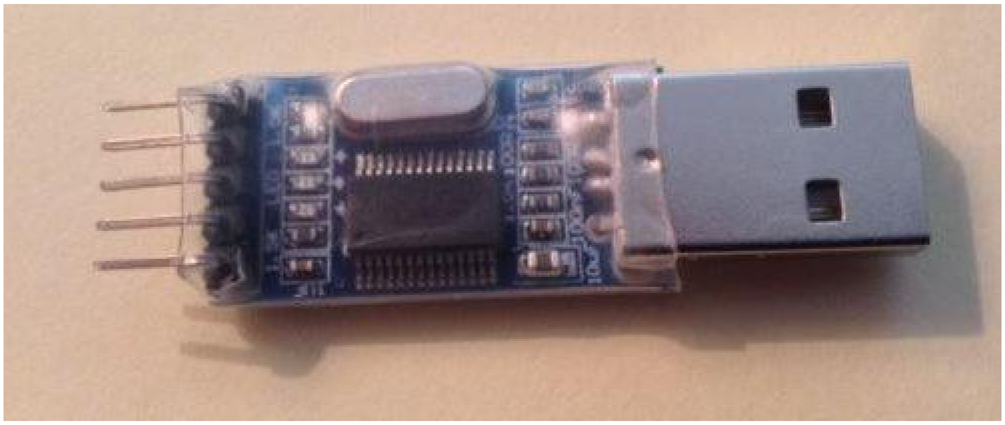
signal	**color**	**disco board (AF1)**
+5v	Red	+5v
Ground	Black	GND
TX	Orange (output)	PA-10 (RX)
RX	Yellow (input)	PA-9 (TX)
nCTS	Brown (input)	PA-12 (RTS)
nRTS	Green (output)	PA-11 (CTS)

USB/3.3v Serial hardware Option 3

- $1 cheap Unix friendly PL2303 chipset usn 3.3v serial dongle (usb3.3v-dongle), but only has TX,RX,0V,+3.3 amd +5V pin.

Note

nCTS and nRTS wires can be soldered onto the PL2303 28 pin SSOP28 chip used in the dongle. These are very fine pitch and it isn’t easy without the right equipment. Theyre also fiddly and fragile so its better to but the Option 1 units above.



PL2303 28 pin SSOP28 Package Pinouts

signal	pl2303 pin
nDTR	2
nDSR	9
nDCD	10
nCTS	11

signal	pl2303 pin
nRTS	3

First Things First, test your serial terminal

The first thing to do is test your serial terminal in loopback, with and without hardware handshaking to make sure it works.

1. OS: FreeBSD 10.1
2. Cable: FTDI usb serial cable (option 1 above).
3. Terminal Program: Picocom

Second, do a Loopback test, with no flow control

Connect the TX and RX wires together and see if the terminal echoes keys typed, if it does then it’s working. If not, possible causes are:- 1) You don’t have permissions to talk to your serial port 2) The wrong serial device is being specified

```
picocom -b 115200 /dev/cuaU1 –imap lfcrlf,crcrlf –omap delbs,crlf
```

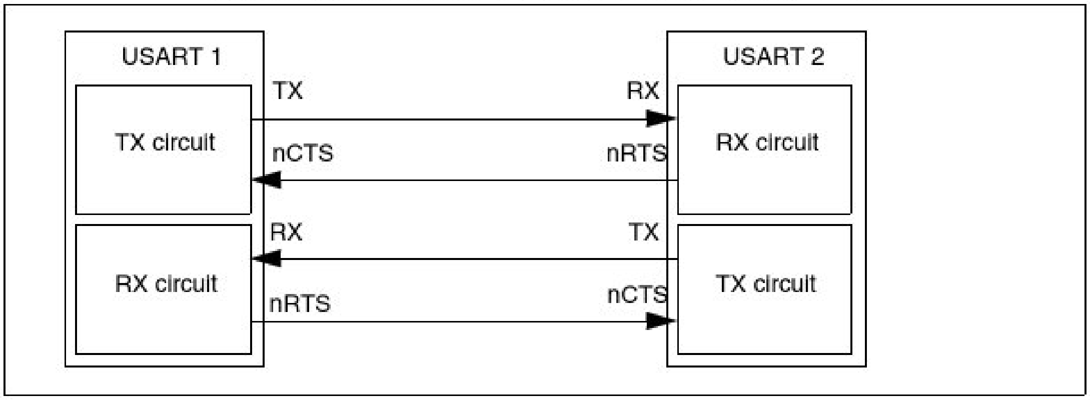
Third, Loopback with hardware flow control

1. When /CTS (brown wire) is connected to 0v (LOW), characters typed should be echoed. When /CTS is HIGH the characters should not be echoed.
2. /RTS (green wire) isn’t used for this test

```
picocom -b 115200 /dev/cuaU1 --imap lfcrlf,crcrlf --omap
delbs,crlf --flow h
```

Handshaking Timing figure

Figure 251. Hardware flow control between 2 USARTs


USART 1
TX circuit
RX circuit
TX
nCTS
RX
nRTS
USART 2
RX circuit
TX circuit
RX
nRTS
TX
nCTS

RTS and CTS flow control can be enabled independently by writing the RTSE and CTSE bits respectively to 1 (in the USART_CR3 register).

STM32F0 discovery board mods for hardware flow control

- Uses USART1
- nCTS: Clear To Send blocks the data transmission at the end of the current transfer when high
- nRTS: Request to send indicates that the USART is ready to receive data (when low).

STM32F0 Pin	**Flow control Function**	**ALT Configuration**
PA12	USART1_RTS	AF1 - 0001

STM32F0 Pin	Flow control Function	ALT Configuration
PA11	USART1_CTS	AF1 - 0001
PA9	USART1_TX	AF1 - 0001
PA10	USART1_RX	AF1 - 0001

RTS and CTS flow control

can be enabled independently by writing the RTSE and CTSE bits respectively to 1 (in the USART_CR3 register).

STM32F0 RTS flow control

If the RTS flow control is enabled (RTSE=1), then nRTS is asserted (tied low) as long as the USART receiver is ready to receive a new data. When the receive register is full, nRTS is deasserted, indicating that the transmission is expected to stop at the end of the current frame. Figure 252 shows an example of communication with RTS flow control enabled.

CTS

Signal	I/O	Description

Signal	I/O	Description
CTS	IN	Clear to send. Indicates whether the UART can start transmitting data when flow control is enabled.
State		– Asserted: Data transmission can start. – Negated: Data transmission cannot start.
Timing		– Assertion When transmitting device’s RTS asserts. – Negation When transmitting device’s RTS deasserts.

RTS

Signal	I/O	Description
RTS	OUT	Request to send. When driven by the receiver, indicates whether the UART is ready to receive data. When driven by the transmitter, can enable an external transceiver during transmission.

Signal	I/O	Description
State		– Asserted: When driven by the receiver, ready to receive data. When driven by the transmitter, enable the external transmitter. – Negated: When driven by the receiver, not ready to receive data. When driven by the transmitter, disable the external transmitter.
Timing		– Assertion Can occur at any time; can assert asynchronously to the other input signals. – Negation Can occur at any time; can deassert asynchronously to the other input signals.

Thinking Forth Tips

Everything old is new again. There are no new ideas. Modern software development is not so modern.

Here are some random tips (quoted verbatim) from the book ThinkingForth (published in 1984):

- Start simple. Get it running. Learn what you’re trying to do. Add complexity gradually, as needed to fit the requirements and constraints. Don’t be afraid to restart from scratch.
- Plan for change (by designing components that can be changed).
- First, and most importantly, the conceptual model should describe the system’s interfaces.
- Factor the fruit. (Don’t confuse apples with oranges).
- You don’t understand a problem until you can simplify it.
- Decide on error and exception-handling early as part of defining the interface.
- Generality usually involves complexity. Don’t generalize your solution any more than will be required; instead, keep it changeable. (hmmm… YouArentGonnaNeedIt? – Ed)
- To simplify, take advantage of what’s available.
- The mean time for making a “two-hour” addition to an application is approximately 12 hours.
- In deciding where to start designing, look for:
    - areas where the most creativity is required (the areas where change is most likely)
    - areas that give the most satisfying feedback (get the juices flowing)

    - areas in which the approach decided upon will greatly affect other areas, or which will determine whether the stated problem can be solved at all
    - things you should show the customer, for mutual understanding
    - things you can show the investors, if necessary for the rent.

Thumb Instruction Set

Note

The THUMB Instruction Set is used throuought all assembly code examples and projects in this site.

The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions. Thumb instructions are each 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect on the processor model. Thumb instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and Thumb states.

On execution, 16-bit Thumb instructions are transparently decompressed to full 32-bit ARM instructions in real time, without performance loss.

Thumb has all the advantages of a 32-bit core:

- 32-bit address space
- 32-bit registers
- 32-bit shifter, and Arithmetic Logic Unit (ALU)
- 32-bit memory transfer.

Thumb therefore offers a long branch range, powerful arithmetic operations, and a large address space.

Thumb code is typically 65% of the size of ARM code, and provides 160% of the performance of ARM code when running from a 16-bit memory system. Thumb, therefore, makes the ARM7TDMI core ideally suited to embedded applications with restricted memory bandwidth, where code density and footprint is important.

The availability of both 16-bit Thumb and 32-bit ARM instruction sets gives designers the flexibility to emphasize performance or code size on a subroutine level, according to the requirements of their applications. For example, critical loops for applications such as fast interrupts and DSP algorithms can be coded using the full ARM instruction set then linked with Thumb code.

`Download THUMB INSTRUCTION SET`

`Download THUMB QUICKREF`

`Download THUMB SUPPLEMENT`

Thumbulator

Thumbulator is a Thumb (16 bit ARM) instruction set simulator by dwelch67
https://github.com/dwelch67/thumbulator

Thumbulator is a simple alternative to QEMU/KVM etc.

Note

A modified Thumbulator is supplied and used with Mecrisp-Stellaris

Matthias makes use of a modified Thumbulator to build the binary image files of Mecrisp-Stellaris complete with Words such as the disassembler for some MCU’s. These files (where built) can be found in in a “PRECOMPILED” sub directory of your device.

At this time of writing the Thumbulator emulates the following devices for mecrisp-stellaris-2.4.7.

Device
aducm4050
lm4f120-tm4c1294
numworks

Device
stm32f051
stm32f103
stm32f303
stm32l053

Run Mecrisp-Stellaris-stm32f051-RA on your PC!

If you have Linux, you can actually run Mecrisp-Stellaris on it right now, without any hardware at all using Thumbulator. Your choice is limited to the above Devices but it’s a easy way to check Forth out. There are virtually no peripherals, one uart port for TX only (prints the character to the console). A port that prints the 32 bit value written to the console, and a port to halt the processor.

I make use of Thumbulator when I need to test different versions of Mecrisp-Stellaris such as compare Inlining between the the RA and the Classic Core for this Documentation Site.

How to:

I’ve used a stm32f051-RA kernel but you can easily change to any in the list above.

Note

FreeBSD and OpenBSD users may have to change a few paths etc and install Bash then symlink it to /bin/bash.

Basically you just download the latest Mecris-Stellaris Release (I’ve used a old release here) and perform the following:

Action	Command
Uncompress	mecrisp-stellaris-2.4.7
cd	/mecrisp-stellaris-2.4.7
execute	release
execute	thumbulator/runcore thumbulator/thumbulator-stm32f051 stm32f051-ra/precompiled/mecrisp-stellaris-stm32f051-with-disassembler-m0.bin

After the last command you should see:

```
Mecrisp-Stellaris RA 2.4.7 with M0 core for STM32F051 by
Matthias Koch
```

Warning

If anything is wrong with your paths or syntax etc, you will get a CORE file in the directory. This is nothing to worry about, just delete the CORE and check the syntax and paths to find the error.

As mecrisp-stellaris-stm32f051-with-disassembler-m0.bin has the dissasembler installed you can create a Word and see the contents in assembly.

```
: 3plus  3 + ;  ok.

see 3plus

20000398: 3603  adds r6 #3
2000039A: 4770  bx lr

ok.
```

Turnkey

Methods to create a final Mecrisp product.

Warning

! UNDER CONSTRUCTION !

Supported MCU’s

```
aducm4050
lm4f120-tm4c1294
stm32f051.c
stm32f103
stm32f303
stm32l053
```

Thumbulator

This method allows the fast creation of a “prebuilt” Mecrisp Kernel Image containing all your Forth project files.

The new project Image can then be flashed to one or more Microcontrollers.

tm4c1294 example

Example Project Directory

```
project/
├── thumbulator/
├── source/
├── tidyup
├── release
```

```
├──
├──
├──
├──
├──
├──





thumbulator/
├── m3partial.h
├── Makefile
└── thumbulator.c

source/
├── 00-helper.fs
├── 00-jtag.fs
├── 01-dict-compiler.fs
├── 02-primitive.fs
├── 03-samples.fs
├── 04a1-target-mem-card.fs
├── 04a2-reg-manag.fs
├── 04a3-routine.fs
├── 04a4-cross-reg-locat.fs
├── 04a5-var-and-buff.fs
├── 04b-stack-prim.fs
├── 04c-single-prim.fs
├── 04d-double-prim.fs
├── 04e-comp-prim.fs
├── 04f-memory-prim.fs
├── 04g-sliding-prim.fs
├── 04h-mult.fs
├── 05-dissasembler.fs
├── 06-cornerstone.fs
├── 07-more-definitions.fs
```

Technique

1. Forth project files are in the project/source and numbered in required load order.
2. Execute “release”

Programming Question And Answers

Answered from the Perspective of Mecrisp-Stellaris Forth.

Editing Words

I’m curious with how you deal with changing the number of items a Word is expecting/leaving on the stack? let’s say you defined a word FOO that expected 2 numbers on the stack, and left 1 number on the stack. Now you change FOO to return 2 Stack items.

Q: How do you check that everything that calls FOO is now ready for the extra Stack item ?

A: I just reload and run the modified program and check the Stack. Here is an example:

Example

Here we define FOO and FOO2X which satisfies the above question.

: FOO (u1 u2 - - u3) + ; add two parameters, return the addition on the Stack

: FOO2X (- - FOO * 2) FOO 2 * ; pass two parameters to FOO then multiply the result by 2 and place the result on the Stack

1 2 FOO2X . .s check the Stack contents, there should be nothing on it.

Uploading the code produces this output:

```
1 2 FOO2X . 6  ok.
.s [0 ] 42  ok.                       \ Ignore the "42" the
Stack counter "[0 ]" shows nothing is left on the stack
```

The Words performed as expected.

Editing Foo

Foo is now edited so that it will return two numbers, the first is the addition of the parameters 1 and 2, the second is the number 2

> : FOO (u1 u2 - - u3) + 2 ; add two parameters, return the addition on the Stack PLUS the number 2, this is our extra Stack item.
>
> : FOO2X (FOO - - FOO 2 *) FOO 2 * ; nothings changed here
>
> > 1 2 FOO2X . .s check the Stack contents, there should be nothing on it.

```
1 2 FOO2X . 4  ok.
.s  [1 ] 42 3  ok.                    \ PROBLEM ! there is a
UNEXPECTED STACK ITEM shown by "[1 ]", it's the number 3
(ignore the 42 again)
```

Something has gone wrong here (as expected) and the clue is that the stack has a number left on it, the number 3 which was the addition of the parameters 1 and 2 given to FOO.

FOO2X was only expecting ONE parameter and was presented with two, so it took the first one the “extra number 2” we added above.

We were not expecting anything on the Stack and finding the Stack is NOT EMPTY is how I handle checking that everything that invokes FOO is now ready for the extra parameters.

VIM

By Bram Moolenaar: https://moolenaar.net/vim.html

Using Vim completion to easily insert bitfield configuration template Words into your Forth programs.

See also

BITFIELDS, DECLARE EARLY ?

Svd2forth

svd2forth derives output from ARM *CMSIS-SVD* files and transforms them using *XSLT*. In one fast operation svd2forth produces a number of files for the chosen STM32Xxxxx MCU.

The file we are interested in here is named *bitfields.fs* and contains complete configuration *templates* for the hardware in your MCU. Why would you want them ?

Note

Bitfields.fs can be likened to the *Include files* used by The C Programming Language, but bitfields.fs also contains the description field plus the Stack Picture.

Bitfields.fs

This is a small sample of the bitfields.fs file produced for the STM32F051 MCU. This segment contains the MODE configuration *template* Words for GPIOC, BITs 0 - 15.

See the **`Technical Manual`**, page 148/1004 for General-purpose I/Os (GPIO) information.

```
\ GPIOC_MODER (read-write) Reset:0x00000000
: GPIOC_MODER_MODER15 ( %bb -- x addr ) 30 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER15, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER14 ( %bb -- x addr ) 28 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER14, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER13 ( %bb -- x addr ) 26 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER13, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER12 ( %bb -- x addr ) 24 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER12, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER11 ( %bb -- x addr ) 22 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER11, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER10 ( %bb -- x addr ) 20 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER10, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER9 ( %bb -- x addr ) 18 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER9, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER8 ( %bb -- x addr ) 16 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER8, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER7 ( %bb -- x addr ) 14 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER7, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER6 ( %bb -- x addr ) 12 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER6, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER5 ( %bb -- x addr ) 10 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER5, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER4 ( %bb -- x addr ) 8 lshift GPIOC_MODER ;
\ GPIOC_MODER_MODER4, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER3 ( %bb -- x addr ) 6 lshift GPIOC_MODER ;
\ GPIOC_MODER_MODER3, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER2 ( %bb -- x addr ) 4 lshift GPIOC_MODER ;
\ GPIOC_MODER_MODER2, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER1 ( %bb -- x addr ) 2 lshift GPIOC_MODER ;
\ GPIOC_MODER_MODER1, Port x configuration bits y =  0..15
: GPIOC_MODER_MODER0 ( %bb -- x addr ) GPIOC_MODER ; \
GPIOC_MODER_MODER0, Port x configuration bits y =  0..15
```

GPIOC_MODER_MODER8 Breakdown

Syntax	**Description**
:	Definition start
GPIOC_MODER_MODER8	Definition Name: Peripheral_Register_Bitfield transformed from CMSIS-SVD
(%bb – x addr)	Stack Picture; %bb means it expects two bits as input parameters. X = some number, addr = a 32 bit address
16	decimal number = 16
lshift	shift left, 16 bits in this case
GPIOC_MODER	32 bit address constant defined in memmap.fs. GPIOC_MODER = $48000800
;	Definition completion
GPIOC_MODER_MODER8	Definition name repeated in comments

Syntax	Description
Port x configuration bits y = 0..15	Description Field, transformed from CMSIS-SVD

A brief code example

This is code from “f0-disco-leds.fs. It is uploaded to a STM32F0 Discovery Board running Mecrisp-Stellaris and “leds-init” is then executed. After that the on-board blue and green leds may be controlled by Words such as “green-on” etc.

```
%00 constant INPUT
%01 constant OUTPUT
%10 constant AF
%11 constant ANALOG

: bit ( u -- u ) 1 swap lshift1-foldable ; \ turn a bit
position into a binary number.

: GPIOC_MODER_MODER8 ( %bb -- x addr ) 16 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER8
: GPIOC_MODER_MODER9 ( %bb -- x addr ) 18 lshift GPIOC_MODER
; \ GPIOC_MODER_MODER9
: GPIOC_BSRR_BS9 ( -- )  9 bit GPIOC_BSRR ! ; \
GPIOC_BSRR_BS9
: GPIOC_BSRR_BR9 ( -- ) 25 bit GPIOC_BSRR ! ; \
GPIOC_BSRR_BR9
: GPIOC_BSRR_BS8 ( -- )  8 bit GPIOC_BSRR ! ; \
GPIOC_BSRR_BS8
: GPIOC_BSRR_BR8 ( -- ) 24 bit GPIOC_BSRR ! ; \
GPIOC_BSRR_BR8

: leds-init ( -- )
  output GPIOC_MODER_MODER8 bis!
  output GPIOC_MODER_MODER9 bis! ;

: green-on  ( -- ) GPIOC_BSRR_BS9 ; \ green led on
: green-off ( -- ) GPIOC_BSRR_BR9 ; \ green led off
: blue-on   ( -- ) GPIOC_BSRR_BS8 ; \ blue led on
: blue-off  ( -- ) GPIOC_BSRR_BR8 ; \ blue led off
```

Summary

- bitfields.fs contains all the Forth bitfield configuration templates for your STM32Xxxx MCU
- each bitfield can be pasted into your code and used in your program as above.
- this method shows the hardware AND software operation in a way that can be read and understood decades later. I wrote about why I prefer this method here.

Inserting a bitfield template into your code

There are two ways that I know of, the easy way and the hard way.

The Hard Way

Have multiple files open, (easy with a tabbed editor) then copy and paste from bitfields.fs into your program.

The Easy Way

Use a capable editor that can search all opened files from your program and insert the template you want.

Vim Completion

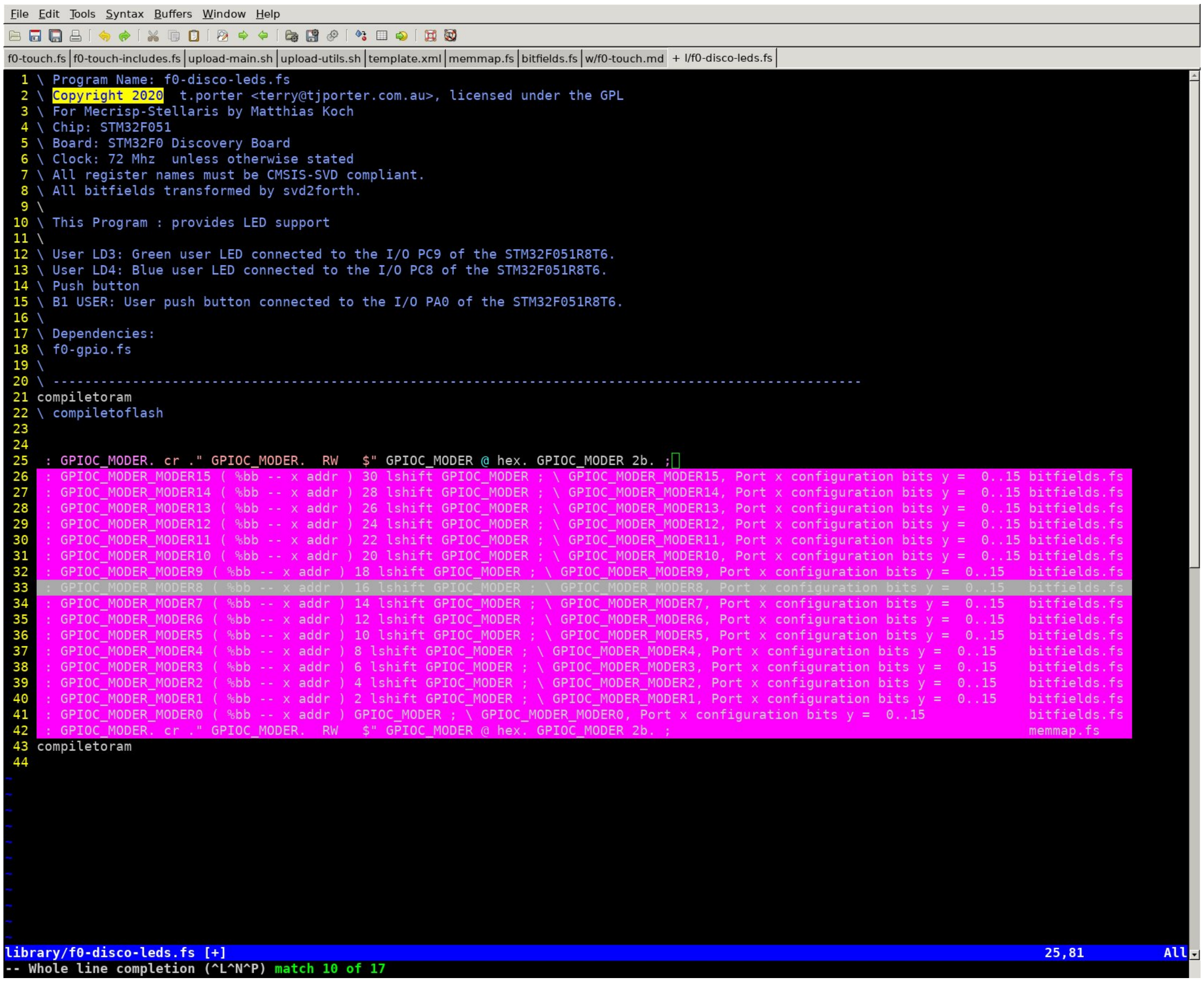
Vim offers many *completion* options and one is described here.

Whole Line Completion

While writing f0-disco-leds.fs I enter (in insert mode):

```
GPIOC_MODER
           ^ cursor is here
<ctrl>x, <ctrl>l
select GPIOC_MODER_MODER8 from the pink coloured pop up
window.
```

When I hit the enter key the line is then pasted into my program. I have found this to be a fast and easy method to locate and insert the bitfield templates needed.


File Edit Tools Syntax Buffers Window Help
f0-touch.fs | f0-touch-includes.fs | upload-main.sh | upload-utils.sh | template.xml | memmap.fs | bitfields.fs | w/f0-touch.md | + l/f0-disco-leds.fs

```
\ Program Name: f0-disco-leds.fs
\ Copyright 2020  t.porter <terry@tjporter.com.au>, licensed under the GPL
\ For Mecrisp-Stellaris by Matthias Koch
\ Chip: STM32F051
\ Board: STM32F0 Discovery Board
\ Clock: 72 Mhz  unless otherwise stated
\ All register names must be CMSIS-SVD compliant.
\ All bitfields transformed by svd2forth.
\
\ This Program : provides LED support
\
\ User LD3: Green user LED connected to the I/O PC9 of the STM32F051R8T6.
\ User LD4: Blue user LED connected to the I/O PC8 of the STM32F051R8T6.
\ Push button
\ B1 USER: User push button connected to the I/O PA0 of the STM32F051R8T6.
\
\ Dependencies:
\ f0-gpio.fs
\
\ ---------------------------------------------------------------------------------------------------
compiletoram
\ compiletoflash


 : GPIOC_MODER. cr ." GPIOC_MODER.  RW   $" GPIOC_MODER @ hex. GPIOC_MODER 2b. ;
 : GPIOC_MODER_MODER15 ( %bb -- x addr ) 30 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER15, Port x configuration bits y =  0..15 bitfields.fs
 : GPIOC_MODER_MODER14 ( %bb -- x addr ) 28 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER14, Port x configuration bits y =  0..15 bitfields.fs
 : GPIOC_MODER_MODER13 ( %bb -- x addr ) 26 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER13, Port x configuration bits y =  0..15 bitfields.fs
 : GPIOC_MODER_MODER12 ( %bb -- x addr ) 24 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER12, Port x configuration bits y =  0..15 bitfields.fs
 : GPIOC_MODER_MODER11 ( %bb -- x addr ) 22 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER11, Port x configuration bits y =  0..15 bitfields.fs
 : GPIOC_MODER_MODER10 ( %bb -- x addr ) 20 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER10, Port x configuration bits y =  0..15 bitfields.fs
 : GPIOC_MODER_MODER9 ( %bb -- x addr ) 18 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER9, Port x configuration bits y =  0..15   bitfields.fs
 : GPIOC_MODER_MODER8 ( %bb -- x addr ) 16 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER8, Port x configuration bits y =  0..15   bitfields.fs
 : GPIOC_MODER_MODER7 ( %bb -- x addr ) 14 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER7, Port x configuration bits y =  0..15   bitfields.fs
 : GPIOC_MODER_MODER6 ( %bb -- x addr ) 12 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER6, Port x configuration bits y =  0..15   bitfields.fs
 : GPIOC_MODER_MODER5 ( %bb -- x addr ) 10 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER5, Port x configuration bits y =  0..15   bitfields.fs
 : GPIOC_MODER_MODER4 ( %bb -- x addr ) 8 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER4, Port x configuration bits y =  0..15    bitfields.fs
 : GPIOC_MODER_MODER3 ( %bb -- x addr ) 6 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER3, Port x configuration bits y =  0..15    bitfields.fs
 : GPIOC_MODER_MODER2 ( %bb -- x addr ) 4 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER2, Port x configuration bits y =  0..15    bitfields.fs
 : GPIOC_MODER_MODER1 ( %bb -- x addr ) 2 lshift GPIOC_MODER ; \ GPIOC_MODER_MODER1, Port x configuration bits y =  0..15    bitfields.fs
 : GPIOC_MODER_MODER0 ( %bb -- x addr ) GPIOC_MODER ; \ GPIOC_MODER_MODER0, Port x configuration bits y =  0..15          bitfields.fs
 : GPIOC_MODER. cr ." GPIOC_MODER.  RW   $" GPIOC_MODER @ hex. GPIOC_MODER 2b. ;                                       memmap.fs
compiletoram

~
~
~
~
~
~
~
~
~
~
library/f0-disco-leds.fs [+]                                                                          25,81          All
-- Whole line completion (^L^N^P) match 10 of 17
```

Vimrc

Some notes on what your ~/.vimrc requires for this to work. These are taken from my ~/.vimrc. Note there are probably a hundred ways to do this.

```
" Omni completion CTRL-X CTRL-O
set omnifunc=syntaxcomplete#Complete

" The default value is .,w,b,u,t,i
"    .: The current buffer
"    w: Buffers in other windows
"    b: Other loaded buffers
"    u: Unloaded buffers
"    t: Tags
"    i: Included files
set complete=b
```

Visualising the Forth Stack

> Visualize - form a mental picture of something that is invisible or abstract.

```
                          -.-.-,~ .
                          )     (
                          |_    |
                          /(_)---`\
                         (_      -'
The Forth Stack is doing  ]      |
my head in man ...        |    _,')
                          [_,-'_-'(
                         (_).-'    \
                         / /        \
```

Note

the Ascii code below is inserted into the Fourth source and NOT commented out for this article. The code won’t run as it is with the Ascii text uncommented. I wrote this page a few years ago and this method definitely helped me visualise the stack but these days I use a very simple method I call “pr”

Forth looks like a Stealth Aircraft

Forth is a very unique language, it’s a bit like a stealth aircraft because when viewed from different angles it looks like something else entirely.

One Forth statement may look a lot like Asembly Language :-

```
01  24 lshift $48000800 bis!
```

Yet, another line of Forth code, where the syntax is now that of the problem it is solving might look totally different, so what’s going on ?

```
: blink-forever begin blink key? until ;                \
blink until a keyboard key is pressed
```

Disruptively Simple

Whilst Forth may look very confusing to the uninitiated, it is actually *disruptively simple* under the covers. Once the ‘aha factor’ kicks in; and the light bulb over your head turns on, you will wonder why you ever had any trouble understanding Forth in the first place. You may even do cartwheels of joy at *discovering* a awesome problem solving programming language that’s been around since the 70’s.

See also

“Starting Forth”, an old classic for beginners : http://home.iae.nl/users/mhx/sf.html

The Stack

The next area of confusion, especially for me, (coming from a hobbyist C and Assembly Language background) was the *Forth Stack*, otherwise known as the *Data*, or *Parameter Stack*.

Non Stack Code

A Forth program which has reached the point where its syntax is that of the problem it has solved can be very easy to read. I bet you get the general idea of what this code is doing ?

For those that don’t, “26 temperature” will light a Magenta colored RGB led.

```
\ composite color table
: white        red-on      green-on        blue-on     ;
: magenta      red-on      green-off       blue-on     ;
: red          red-on      green-off       blue-off    ;
: yellow       red-on      green-on        blue-off    ;
: cyan         red-off     green-on        blue-on     ;
: green        red-off     green-on        blue-off    ;
: blue         red-off     green-off       blue-on     ;
: off          red-off     green-off       blue-off    ;

\ temp selection 10 C to 27 C as we only have 7 colors

: temperature ( temp -- color )
   dup 0  = if     off     else
   dup 10 < if     blue    else
   dup 13 < if     green   else
   dup 16 < if     cyan    else
   dup 19 < if     yellow  else
   dup 22 < if     red     else
   dup 25 < if     magenta else
   dup 28 < if     white   else
   ." ---- error, temp is above 28 C " cr
   THEN THEN THEN THEN THEN THEN THEN THEN
;
```

Stack Code

A more complex program involving multi Stack depths is another issue entirely.

The following Forth program (Minmax) takes data from an array containing X analogue to digital converter samples and produces the minimum, maximum, noise and average values like so:

Minmax Output

```
-----------------------------------------
mimnax code output for input: "800 minmax"
-----------------------------------------

data, minv and maxv = 1215 loop:799
data = 1215 loop:798
data = 1218 (new maxv) loop:797
... lots of lines of data
data = 1215 loop:1
data = 1217 loop:0
data = 1216 loop:-1

Total: 800 data points in mV
Noise = 16 mV
Max     = 1225     (Max - 1/2 Noise) = 1217
Min     = 1209     (Min + 1/2 Noise) = 1217
Average = 1220

ok.
```

Minmax Source Code

This is Minmax in all its Forthy glory, can you understand what the Stack is doing, or is that a headache you feel coming on ?

```
------------------
Minmax Source Code
------------------



: index? ( index? -- element ) raw-adc-data SWAP cells + @ ;
\ fetch array elements

0 variable maxv
0 variable minv
0 variable counter
0 variable average
```

```
: minmax ( array elements -- min max average ) 1 - dup >r
counter ! cr cr
   counter @ index? dup dup dup minv ! maxv ! average ! ."
data, minv and maxv = " .   \ save last array element into
maxv and minv
       begin
           counter @ dup dup  ." loop:" . cr 0  >=  while
\ index 0 = loop -1
           index? dup dup dup dup dup average @ + average !
." data = " .
               minv @ < if
               minv ! ." (new minv) "
                   else        drop
\ data > minv so it's ignored
                   maxv @ > if
                   maxv ! ." (new maxv) "
                   else drop
\ data < maxv so it's ignored
                   then
               then
               counter @ 1 - counter !
       repeat
   drop
   cr
   ."  Total: " r@ 1 + . ." data points in mV" cr
   ."  Noise = " maxv @ minv @ - dup . 2 / dup dup ." mV" cr
   maxv @ dup ."  Max     = " . ."     (Max - 1/2 Noise) = "
swap - . cr
   minv @ dup ."  Min     = " . ."     (Min + 1/2 Noise) = "
swap + . cr
   ."  Average = "  average @ r> / . cr cr
;
```

Now lets insert some Stack Diagrams into the code to help visualise where the data goes as it is ‘popped’ off the Stack. This way you can account for every DUP and every DROP quite easily, compared to holding them in your head as you debug the program.

Minmax with Stack Diagrams

Is this code any clearer, do you get some idea where data pushed onto the stack is going ?

```
-----------------------------------------------------
Minmax source code with embedded Asciio Stack Diagrams
-----------------------------------------------------

: minmax ( array elements -- min max average ) 1 - dup >r
counter ! cr cr
    .------------.
    |   Stack    |
    |------------|
    | counter -1 |-----> >r
    | dup        |-----> counter !
    '------------'
   counter @ index? dup dup dup minv ! maxv ! average ! ."
data, minv and maxv = " .     \ save last array element into
maxv and minv
    .--------------.
    |  Data Stack  |
    |--------------|
    | counter valu |-----> minv !
    | dup          |-----> maxv !
    | dup          |-----> average !
    | dup          |-----> ." data, minv and maxv = " .
    '--------------'
   begin
       counter @ dup dup  ." loop:" . cr 0  >=  while
\ smallest index 0 = loop -1
    .------------------.
    |  data stack      |
    |------------------|
    | new counter valu |-----> ." loop:" .
    | dup              |-----> 0 >=  while------.
    | dup              |                        'index? or
DROP after repeat
    '------------------'
       index? dup dup dup dup dup average @ + average ! ."
data = " .
    .--------------.
    |  Data Stack  |
    |--------------|
    | next element |-----> average @ + average !
    | dup          |-----> ." data = " .
    | dup          |-----> minv @ < if--.
    | dup          |                    '------> minv ! ."
(new minv) " or DROP after ELSE
    | dup          |-----> maxv @ > if---.
    | dup          |                     '---->  maxv ! ."
(new maxv) " or DROP after ELSE
    '--------------'
       minv @ < if
       minv ! ." (new minv) "
       else    drop
\ data > minv so it's ignored
           maxv @ > if
           maxv ! ." (new maxv) "
           else drop
\ data < maxv so it's ignored
           then
       then
       counter @ 1 - counter !
\ Decrement counter
   repeat
   drop
   cr
   ."  Total: " r@ 1 + . ." data points in mV" cr
   ."  Noise = " maxv @ minv @ - dup . 2 / dup dup ." mV" cr
   maxv @ dup ."  Max     = " . ."     (Max - 1/2 Noise) = "
swap - . cr
   minv @ dup ."  Min     = " . ."     (Min + 1/2 Noise) = "
swap + . cr
             ."  Average = "  average @ r> / . cr cr
;
```

This code includes Stack Diagram text boxes made using Asciio. It has made a big difference helping me visualise the Forth Stack, perhaps it may do the same for you ? (note: the text boxes are not commented-out for clarity).

I don’t leave the text stack diagrams in my code, they’re only used during design and debugging.

```
         ,       ,
         \\_    /|
         /- _`-/ '
        (/\/ \  /\
        O O   ) / |
        `-^--'`<  '
       (_)   _  )/
        `.___/` /
          `--' /
<---.   __ / __ \
<---|==(fl)=) \ /===
<---'   `-' `._,'\
           \     /
            ( ( / \__
         ,---_' |    \
         `-(____)    V

Asciio on FreeBSD RULES!
```

wordlists, vocabularies, vocabulary prefixes and implicit context switching

A big *Thank You* to [manfredmahlow] for creating *Vocabularies* for Mecrisp-Stellaris !

Software Versions used in this demo

- The latest Mecrisp-Stellaris https://sourceforge.net/projects/mecrisp/files/mecrisp-stellaris-2.3.9a.tar.gz is required to use Manfred Mahlow’s VOCS which is included in the common/experimental/vocs-0.7.0 directoru of the release.
- Howto’s include :-
    - VOC-HOWTO-Classes-1: Shows some basics of VOCs based OOP.
    - VOC-HOWTO-Classes-2: How to create classes, using instance variables, defined in other classes.
    - VOC-HOWTO-DataTypes: A short demo to show how VOCs can be used to define data types.
    - VOC-HOWTO-Registers: Using VOCs to implement register identifiers.

VOC-HOWTO-Registers

```
\ VOC-HOWTO-Registers
MM-170618
\ -------------------------------------------------------------------
-----------
\              HOWTO for Mecrisp-Stellaris for VOC Version 0.6.2-FR

\     A short demo to show how to use VOCs to define register
identifiers.

\                  Copyright (C) 2017 Manfred Mahlow

\        This is free software under the GNU General Public License
```

```
v3.
\ -------------------------------------------------------------------
-----------

  key drop

{
  Modern MCUs have lots of peripheral units to be controlled with
many register
  identifiers. In Forth this identifiers are often implemented as
constants,
  cluttering the dictionary.

  With good hardware designs, the register identifiers should be a
combination
  of a unit specific base addres and a register specific offset
address. Then
  they can be implemented as such, significantly reducing the number
of
  required words.

  But earlier or later name conflict may arise when different units
use the
  same register name for a different offset address.

  Using VOCs to implement register identifiers can avoid such
conflicts and can
  significantly reduce the number of identifiers to be implemented.
}

  key drop

  compiletoram

\ Loading vocs.txt    ** Please first read the preamble in that file.
**
\ -------------------------------------------------------------------
----------
  key drop

\ #require das.txt

#require vocs.txt

  key drop


\ Some tools used:
\ -------------------------------------------------------------------
----------
\
\  ??     Displays the top wordlist of the search order. In the
context Forth
\         the core words are ignored. To also see the core words use
???.
\
\  ..     Returns from a VOC context to the Forth context.
\
\ --------------------------------------------------------------------
----------
  key drop

  ??

\ lfa: = Address: , xt: = Code:  in the original Mecrisp Listing


\ First we create a VOC and words to define unit and register
identifiers
\ --------------------------------------------------------------------
----------
  key drop


  forth definitions

  voc GPIO  GPIO definitions  hex

  : port: ( "name" base-addr -- ) ( GPIO ) casted constant ;

  : reg: ( "name" offset-addr -- ) <builds , does> @ or ;


\ then we define the required register identfiers as members of the
GPIO VOC
\ --------------------------------------------------------------------
----------
  key drop


  $3FC GPIO reg: DATA ( Ein- und Ausgaberegister )
  $400 GPIO reg: DIR  ( Soll der Pin Eingang oder Ausgang sein ? )
  $500 GPIO reg: DR2R ( 2 mA Treiber )
  $504 GPIO reg: DR4R ( 4 mA )
  $508 GPIO reg: DR8R ( 8 mA )
  $50C GPIO reg: ODR  ( Open Drain )
  $510 GPIO reg: PUR  ( Pullup Resistor )
  $514 GPIO reg: PDR  ( Pulldown Resistor )
  $518 GPIO reg: PORTA_SLR  ( Slew Rate )
  $51C GPIO reg: PORTA_DEN  ( Digital Enable )
  $420 GPIO reg: PORTA_AFSEL ( Analog function select )
  $528 GPIO reg: PORTA_AMSEL ( Analog Mode Select )
```

```
\ and finally we define the required unit identifers in the Forth
context.
\ -------------------------------------------------------------------
----------
  key drop


  forth definitions

  $40004000 GPIO port: PORTA
  $40005000 GPIO port: PORTB
  $40006000 GPIO port: PORTC



\ Only the VOC and the unit identifiers are visible in the Forth
context.
\ -------------------------------------------------------------------
----------
  key drop

  ??


\ All unit identifiers share the same set of register identifiers
\ -------------------------------------------------------------------
----------
  key drop

  PORTA ?? .. drop

  key drop

  PORTB ?? .. drop

  key drop

  PORTC ?? .. drop

  key drop

\ and the combination of unit and register name returns the register
identifier.
\ -------------------------------------------------------------------
-----------
  key drop

  PORTA DATA .
  PORTB DATA .
  PORTC DATA .
  PORTA DIR .
  PORTB DIR .
  PORTC DIR .

\ The GPIO implementation uses 644 Bytes.
\ Creating the same register identfiers as constants uses 1292 Bytes.
\ ---------------------------------------------------------------------
-----------
\ Done. Please also see VOC-HOWTO-Classes-1.txt
\
\ Last Revision: MM-170709  voc.txt --> vocs.txt (v0.6.2)
```

VOC Moder Demo

This shows how *vocabularies* easily enable the use of phrases such as ” GPIOA MODER9 ? ” as a alternative to non vocabulary words like ” GPIOA-MODER9? ” along with the economy of OOP and a possible 50% reduction in Flash memory usage for Peripheral and Register definitions.

- A stm32F0 GPIO has eleven+ registers that configure and control it.
- The one which controls whether a GPIO bit is IN, OUT, SPECIAL FUNCTION or ANALOG is named the ‘MODER’ register , which means ‘MODE REGISTER’.
- Each GPIO pin (or bit) in MODER is controlled by a 2bit pair, as shown in the list below.

```
gpioa. GPIOA_MODER (read-write) $28280000
15|14|13|12|11|10|09|08|07|06|05|04|03|02|01|00
00 10 10 00 00 10 10 00 00 00 00 00 00 00 00 00
```

These are the 2bit GPIO MODES (% = binary base) as used in the GPIOA MODER REGISTER above:-

```
%00 constant IN          ( INPUT MODE )
%01 constant OUT         ( OUTPUT MODE )
%10 constant ALTFUNCT    ( ALTERNATE FUNCTION MODE )
%11 constant ANALOG      ( ANALOG MODE )
```

Each GPIO port is 16 bits (GPIOA0 - GPIOA15), and each bit requires a 2bit code in the MODER REGISTER, making the MODER REGISTER 32 bits long.

Using Manfred’s ‘Vocabularies’. The GPIOA MODER register may be read as below. Note: this is a working Mecrisp-Stellaris system and some bits are already set and must not be changed. This all happens in real time in the MCU.

```
GPIOA MODER @ bin. 101000001010000000000000000000  ok.   ( '@' means
'fetch' and 'bin.' means 'display the result in binary' )
```

Set GPIOA BIT 1 to ANALOG MODE (%11) ..

```
ANALOG GPIOA MODER1 !  ok  ( "!" means store )
```

Check GPIOA MODER to see what the new value is …

```
GPIOA MODER @ bin. 101000001010000000000000001100  ok.
```

The GPIOA MODER1 value is also easily read ..

```
GPIOA MODER1 @ . 3  ok.
```

Or use the alternate “?” to display the MODE (my favorite)

```
GPIOA MODER1 ? %11  ok.
GPIOA MODER0 ? %0  ok.
```

Bingo, GPIOA BIT 1 is now in ‘ANALOG’ MODE (%11) and all other bits are unchanged.

Now change GPIOA BIT 1 to INPUT MODE (%00) ..

```
IN GPIOA MODER1 !  ok
```

Check the change occured ..

```
GPIOA MODER @ bin. 101000001010000000000000000000  ok.
```

Or ..

```
GPIOA MODER1 ? %0  ok.
```

GPIOA BIT 1 is now reset to %00 (INPUT MODE) and the other bits are unchanged.

Note

Forth is often criticized for being terse, cryptic or hard, but if we compare our port MODE syntax with that of Arduino (C language), is Forth really harder, perhaps it’s easier ?

Forth - Arduino Comparison, INPUT

Set a Port Pin to INPUT MODE	Code	Comments
Forth on STM32F0	IN GPIOA MODER1 !	Set GPIOA bit 1 to INPUT. When the <enter> key is pressed this command will instantly take effect on the target hardware.
Arduino	pinMode(pin, INPUT);	set pin to INPUT. ‘pin’ can have a value of 0 - 13. The code must first be compiled and then uploaded to the target.

Forth - Arduino Comparison, OUTPUTS

Set all PORT pins to OUTPUTS	Code	Comments
Forth on STM32F0	%01010101010101010101010101010101 GPIOB MODER !	Easy, GPIOB BITS 0 - 15 are now all OUTPUT.

Set all PORT pins to OUTPUTS	**Code**	**Comments**
Arduino	int a; for(a=0;a < 13;a++){ pinMode(a, OUTPUT); }	Which port is ‘pin’ ? On the UNO, ‘pin’ has a value of 0 - 13 and is spread across Ports B and D, so if you want them all to be OUTPUTS, then code like this is required, or you have to refer to the databook and configure each pin individually.

XON/XOFF

For those wondering about using Xon/Xoff for terminal flow control. I tried the method below in Jan 2020 and it didn’t provide useful flow control with my USB/3.3v dongle, exactly as Matthias predicted. It all worked fine right up until the small UART buffers in my STM32F103 microcontroller were overrun.

~ BIG FAIL~

Note from Matthias:

The XON/XOFF mechanism is fine in theory, but often enough, there are buffers quite large in the USB-serial bridge, and the XON/XOFF characters reach the PC for effect when the small UART buffers in microcontroller have been already overflown far beyond their capacity. I think the RTS/CTS handshake is the better choice.

XON and XOFF should be inserted into quit, I think (untested by Matthias, tested by myself):

```
: handshake-quit ( -- )
  begin
    xon
    query
    xoff
    interpret
    ."  ok." cr
  again
;

' handshake-quit hook-quit ! quit
```

YouTube Videos about Forth

Videos

No	YouTube URL	Brief Description
1	https://www.youtube.com/watch?v=dvTI3KmcZ7I	Outstanding Mecrisp-Stellaris intro using a STM32F103 powered “Blue-Pill”

Note

Video 1: Around 7:00 the author states that he doesn’t like the STM32F Datasheet, and for the STM32F103 I agree, the GPIO configuration is very confusing, but remember, this was their FIRST ARM chip and was 10 years old back in 2017. Compared to this chip, the STM32F0xx GPIO is a breeze to configure. He also tries to enable the GPIO (and uses the wrong address) but with Mecrisp-Stellaris ALL THE GPIOS are enabled by default so you don’t need to do this. The GPIO’s are mostly set to the default INPUT mode, apart from some Alternate Function pins such as a USART used for Terminal communications.

My Videos

My YouTube “Channel”

https://www.youtube.com/c/TerryPorterElectronics

No	YouTube URL	Brief Description
11	https://youtu.be/xpTvd1Zhe0U	STM32F103C8 Diagnostics. Did your Blue Pill get a FREE 64kB Flash for Xmas ?
10	https://www.youtube.com/watch?v=60sNQISzEks	Compiling 583 forth programs on a “blue pill” in 23 seconds
9	https://www.youtube.com/watch?v=uWykNpPGsAg	Clone a stm32f103 chip running Forth
8	https://www.youtube.com/watch?v=JuW_noOexng	Stm32f103 forth devkernel showing realtime speed
7	https://www.youtube.com/watch?v=vAGTKLZq0PM	How to add error coloring to mecrisp-stellaris forth

No	YouTube URL	Brief Description
6	https://www.youtube.com/watch?v=UpIYeZxftcE	Temperature sensing with forth and two lmt01 ttemperature sensors
5	https://www.youtube.com/watch?v=od4pkkT9sS8	Cloning a mecrisp-stellaris forth image
4	https://www.youtube.com/watch?v=7DfeEcU-tAs	Embedded forth language on arm based stm32f0 - ide error handling
3	https://www.youtube.com/watch?v=Y7BjuqQnCH8	The blue-pill development board and nokia 5110 lcd scam
2	https://www.youtube.com/watch?v=0gGkwmRNnGA	Stepper driver demo
1	https://www.youtube.com/watch?v=kDi-NIz3-QA	Embedded forth language on arm based stm32f0 - the ide

Index

Symbols | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y**

Symbols

- +loop
- .s
- 12bitencoding
- 28 byte assembly blinky
- 3.3v serial
- 48mhz
- 48mhz clock
- 5110 lcd
- 72mhz
- <builds does>, [1]
- >

A

- about mecrisp-stellaris forth
- about this site
- adc, [1]
- adding a different mcu
- aducm4050
- afr
- alibaba
- aliexpress
- alternate functions
- alternatives
- analog
- analog to digital 19ch scanner
- ansi escape codes
- antiforth by jeoff fox
- apm32f103
- arduino
- arm-none-eabi-as
- arrays, [1]
- Asciio
- assembler, [1]
- assembly code, [1]
- assembly language, [1], [2]
- assembly:blinky code
- assembly:interactive
- asyncronous message passing
- atmel sam: non portable
- atomic, [1]
- atsame70q21
- autobaud

B

- balancing the return stack, [1]
- baudrate, [1]
- baudrate calculator
- beginner
- beginner literature
- benchmark
- benchmarks
- big endian
- bit testing
- bit@
- bitfields, [1], [2], [3]
- bitfields and structures
- bitfields why declare early ?
- bits, [1]
- bits:clearing
- bits:manipulation
- bits:resetting
- blinkies
- blinky, [1]
- blocks
- blue pill, [1], [2], [3]
- blue pill: usb
- blue pill:starter kit
- blue-pill
- bluepill, [1], [2], [3]
- bluepill examples
- bluepill:128kb ?, [1], [2]
- boards
- books
- boot pins
- Bram Moolenaar
- bridge
- buffers
- bugs, [1]

C

- C
- c
- C and forth examples
- c programming language, [1]
- calculator
- calltrace tool
- case
- changelog
- charles moore
- common programs
- comms
- comparator
- comparator:window mode
- compiler essentials
- compiling
- configuration
- configurations: kernel
- configure: external interrupt

chip pictures
chip specials
chips
classes
classic kernel
classic-kernel
clear and set
clock
cloning
cloning-integrated
cmis-svd
cmsis-svd, [1], [2], [3]
code review
color, [1]
colour
colour prompt
colour:terminal
colourized error messages
common problems
constant folding
contributing code
control structures
cooperative multitasking
Copyright
cornerstone
cortex-m
counted string routines
counterfeit
create does>
create...does>
critical systems
cs32f103
custom binary or hex image file
cy8c4245axi
cy8c4247azi
cy8c4248bzi
cy8c5868axi
cy8c5888lti
cypress psocs: non portable

D

data processing
data stack
data types
data:stack
debouncing
debugging, [1]
decision making
deep insights
defining words, [1]
definite loops
delay, [1]
diagnostic
diagnostics
dict
dictionary, [1]
dictionary expansion
dictionary: common hardware access
disassembler
discovery
documentation
documentation:mcu

delay:pure forth
dev board
development
environment
devtips
df-util
Dot
double number math
double-comparisons
dslite2forth
dtc: direct threaded code
dump calltrace, [1]

E

e4thcom
efm32gg990f1024
efm32hg322f64
else
elua
embedded configuration
endianess
error detection, [1]
error messages
error-message:jump too far
example boards
exception, [1]
exception handling, [1]
execute assembly code
execute machine code
exti

F

f0 Discovery Board
f0 discovery board, [1]
f0 low power
f0-id
fake STM32F103 chips
faq
fast ram
fe
file system
fixed point
fixed point arithmetic
fixed point math insights
flag codes
flash: full warning
flashing, [1]
Flowcharting
folding, [1], [2]
follie
forth
forth alterntives
forth explorer
forth intro
forth inventor
forth learning materials
forth philosophy
forth primer

flags
flags and inventory
flash
flash memory
flash size data register
flash. memory
flash: change size
forth versus other systems
forth wizard
forth:blinky code
fossil
free flash
free memory
free ram, [1]
free software

G

garbage collection
garbage on the stack
gas
gd32f103
gd32vf103, [1]
gd32vf103Cx
gdb
geoff fox
glossary
gnu assembler
gnu screen, [1]
gnuscreen:slowpaste
gpio
gpioa, [1]
gpiob
gpiob_afr
gpiox_moder
gpl, [1]
Graphviz

H

hackaday
hacking your own mcu
handshaking. serial
hardware flow control
hardware support
hex file
hooks
html

I

ide, [1]
if
inner interpreter
input

ihex, [1]
imm
implicit context switching
include files
include forth files in binary image
inheritance
init
init word
inline machine code
inlining, [1]
instrument your mcu
integrated circuits
interactive assembly
interrupt, [1]
interrupt example program
interrupt flowchart
interrupts
intro
irq-fault
isr

J

j.l. bezemer
jewel of forth
jtag
jump too far, [1]

K

keil
kernel
kernel: rebuild. kernel: compile
kernel:configuration options
kl25z128vlk4, [1]
kl46z256
known issues

L

learning
leo brody
levels of code in forth programming
libraries
library
License
license
local variable
lockups
logic
loop
loop unrolling
loops
lpc1114fbd48
lpc1114fn28

list
lm4f120h5qr
lm4f232h5qc
lmt01 temperature sensor
loadmeter
lpc1115fbd48
LPC845M301
lpds
lshift

M

machine code, [1]
machine code:inline
make your own
mannipulation: stack
maths
maths library
maths:inverse trig functions
maths:square root
maths:trig functions
max
maximum ram
mbed
mbed:blinky code
mcu not listed ?
mcu: name change
mecrisp-stellaris
mecrisp-stellaris on your pc
memory access
memory mapping
memory size data register
memory status
memory usage
memstat
micropython
millis
millisecond delay
millisecond interrupt
milliseconds
min
min max window
miscellanious
mk20dx256vlh7
mk64fn1m0
moder, [1]
morse code generator
msp430, [1]
msp432p401r
multitasking
multitasking:delay

N

new projects
nuc123

new to forth ?
nokia 5110 lcd display
non portable mcu's
none-arm-eabi
nrf51822
nrf52832
nucleo
nucleo-stm32l072
number bases
numworks calculator, [1]
nvic
nvic_iser0_setena
nvic_iser1_setena

O

olimex p103
olimex p103: usb
olimex-p103
openocd, [1]
operand stack
outer interpreter
output
overclock

P

patches
pc
pc-relative jumps
pdds
pearl of Forth
peripherals
philae lander
picocom
picocom: hardware handshaking 660800 baud
pictured numerical output
pill.green pill
pin connections serial
pioa_afr
pitfalls
porta
portb
porters guide
porting guide
postpone
pr
printing
printing without spaces
problems, [1]
programming language benchmarks
programs
projects
prototyping

Q

qemu
quickstart

R

r
r@
ra
ra v/s classic kernel
ra-kernel, [1]
ra:example
ra:register map
radeus-8200
radiation hardened cpu
ram, [1], [2]
ram: change size
ram: full warning
random access memory
rant:reinventing the wheel
rant:the library myth
rants
readme
recognisers
references
register allocator
register legends
register pack
registers
repl
return stack
return:stack
risc-v
riyadh airport
rosetta spacecraft
rts/cts handshaking
rtx2010 cpu

S

s31.32, [1]
saving
scheme
scm
scr
scs
see
serial, [1]
serial bootloader
stm32f1 usb
stm32f100rb
stm32f103, [1], [2], [3]
stm32f103C8
stm32f103C8 v/s gd32vf103Cx
stm32f103c8: hidden extra 64kB flash

serial terminals
Shenzhen LC-mini STM32F103C8T6-board
signed-comparisons
sigrok
single number math
single-comparisons
sleep mode. stop mode: vreg on
sleepdeep
small endian
smudge
source code errors
source documentation
spacecraft
spaces
special chip
special function
speed of development, [1]
split intiger to digits
square wave generator
st-link
stack, [1], [2], [3], [4], [5]
stack check
stack pointers
stack sensitive prompt
stack visualisation
stack:empty ?
stack:manipulation
stack:printing
stacks
stair-stepping
STM32F103C8:2 x 64kB flash?, [1]
stm32f103c8:diagnostics
stm32f103c8t6
stm32f103c8t6:2 x 64kb flash?
stm32f103xx-diagnostics
stm32f207zg
stm32f303k8
stm32f303re
stm32f303vct6
stm32f401cd
stm32f401re
stm32f407vgt6
stm32f411ret6
stm32f412vgt6
stm32f429zit6
stm32f746ng
STM32G031F8
STM32G031J6
STM32G071RB
stm32l031k6
stm32l053c8t6
stm32l073rz
stm32l082kz
stm32l0xx
stm32l152re
stm32l162rd
stm32l432kc
stop mode: vreg off
storage space
strengths
string formatting
strings
structs
structures

standby mode
stc: subroutine threaded code
stephen pelc
stepper motor
stlink
stm32f0 discovery board
stm32f030f4
stm32f030r8
stm32f042f6
stm32f051
stm32f051r8
stm32f072rb
stm32f0xx, [1], [2]
supported hardware
svd
svd2forth, [1], [2]
svd2forth-v1
svd2forth-v2, [1]
svd2forth-v3, [1]
svd2gas
svd2mecrisp
swap dragon prize
swd
sym32f103
system control register
systick, [1], [2], [3]

T

telnet
temperature sensor project
temperature:sensor simple
terminal
terminal corrupted
terminal hooks
terminal-io
test
testing
the blinky bazar
the first rule of forth
the philosophy of problems
then
thinking forth
thumb
thumb:instruction set
thumbulator
tick word
timer
tiva connected launchpad
tm4c123gh6pm
tm4c1294ncpdt
tm4c129x
todo list
toggle
touch
touch sensor
traffic lights
traps
tsc
tsc peripheral
ttm richter

turnkey, [1], [2], [3]

U

uart handlers
umax
umin
undefined behaviour
Unhandled Interrupt 00000003
unhandled interrupt 00000003
Unhandled Interrupt 00000003 !
Unhandled interrupt 00000003 !
unused dictionary space
uploading sourcecode, [1]
usart1_rx
usart1_tx
usb
usb bridge
usb serial
usb-3.3v Dongle or cables
user input and interpretation
user pushbutton
user question and answers
user submitted code
utube

V

videos, [1]
vim
virtual mecrisp-stellaris
voc moder demo
voc-howto-registers
vocabularies, [1]
vocabulary prefixes
VOCS
vocs

W

wakeup
weaknesses
wfi
what's new?
wierd screen
word list
word usage examples
wordlists

words4

X

xmc1100

xon/xoff

xor

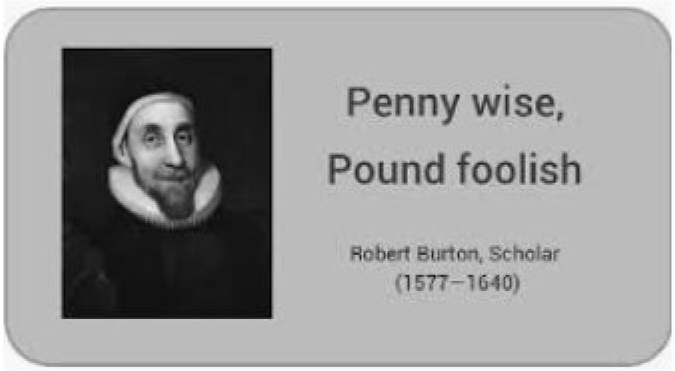
Y

you-tube

youtube

youtube videos, [1]

Three Cent Mcu’s



I’m beyond gobsmacked when I see the fervor of hobbyists salivating over some Chinese OTP mcu with 1k of ROM, a couple of timers and a usart.

Because it’s advertised as costing 3 cents a unit.

If one is making 1 million devices and this device will do, then there is a *tiny* chance that it may be useful but I doubt it because I’m sure STM would offer great pricing on a 16kB STM32F030 variant, or a STM8 or whatever if you order a million units. So would Microchip, Ti etc.

Any company set up to manufacture those kinds of volumes will also have their authorized suppliers and procedures, and the Three Flying Geese manufacturer of 3 cent MCU’s probably won’t be on it.

So that just leaves hobbyists, and even $100 per MCU is no big deal for them because they will probably buy 2 a year. And one gets a damn fine development board with everything for $100, which makes learning fun and easy compared to half baked, incomplete, vendor tool locked junk.

And anyone who thinks $100 is expensive obviously doesn’t know that the Motorola 8 bit MC6800 CPU cost $175 USD each and in the 1970’s that was a lot of money. When the $25 Dollar 6502 bombshell hit the market, the 6800 was quickly reduced to $69, and eventually to $39.

But then one doesn’t need to spend $100 a chip. As I’ve mentioned here before, I bought 480x STM32F051’s with 64kB flash and 8kB ram in 2014 for $0.56 USD each, retail. That’s only $268 for a lifetime hobbyist supply of these Cortex-M0 MCU’s. They are sealed in bags filled with nitrogen gas and include moisture sensing tags, probably not needed for 32 pin UFN’s which have solid tinned copper pins underneath. No clones/fakes in that kit, straight from Avenet in Dallas.

STMicro also have free and easily downloadable documentation for *everything* down to the smallest detail, and in English along with APP notes and code examples etc. So does Microchip, NXP, Texas Instruments, Nordic and a heap of proven, reliable companies.

During our lives we will have virtually unlimited money if we are smart, but no matter how smart we are, we only have limited time available. Why squander it ?

BIT TESTING

Bit testing is a major component of embedded programs, here we examine the default bit testing Words available in Mecrisp-Stellaris and how best to deploy them.

WORK
IN PROGRESS

STM32F103C8, 2 x 64kB flash

Bluepill Blinky

And a gentle introduction to Forth.

If you use the STM32F103C8 DIAGNOSTICS image and have wondered about what Forth is and what it can do (apart from make cool standalone binary diagnostcs programs for Blue Pills) this page may be for you.

This blinky runs on the STM32F103C8 DIAGNOSTICS image. The Bluepill LED is connected to GPIOC-13.

Download

https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/f103-diagnostics-image-blinky.fs/download

The STM32F103C8 DIAGNOSTICS image was made to test Blue Pill boards for hidden extra 64kB Flash plus read chip ID etc, but it’s also a complete working Forth which can do all kinds of cool things.

Blinking a LED is one of the simpler things this image can do.

All you have to do is load the f103-diagnostics-image-blinky.fs file into your Blue Pill and the LED should begin flashing.

From your serial terminal select f103-diagnostics-image-blinky.fs and upload it to the Blue Pill. When the file finishes uploading the LED should begin flashing and after 10 flashes, you will be returned to the terminal prompt.

Some common issues

Lockups

Symptom: The f103-diagnostics-image-blinky.fs upload only goes as far as a few characters and then appears to lock up. Only pressing the Blue Pill reset button fixes it. See this page.

Note

For the 75Mhz STM32F103C8 DIAGNOSTICS image you are running on your Blue Pill, 100ms EOL delay should be plenty.

LED does not blink

Symptom: The program runs and then after 10 seconds it stops and control is returned to the terminal, but the LED doesn’t blink ?

Why? On your board, the LED may not be connected to GPIOC-13. Find out where it is and edit the f103-diagnostics-image-blinky.fs source code or connect a external LED with 510 Ohm Resistor between 3.3V and GPIOC-13.

Source Code

```
\ Program Name: f103-diagnostics-image-blinky.fs
\ Date: Wed 5 Feb 2020 13:33:55 AEDT
\ Copyright 2020 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV2
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Board: Blue Pill with LED on PC-13
\ Designed to be used with STM32F103C8-DIAGNOSTICS.bin
\ https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/STM32F103C8-DIAGNOSTICS.bin
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\ Standalone: no preloaded support files required
\
\ This Program : Blinks the Blue Pill LED on PC-13
\
\ The Blue Pill LED anode is connected to +3.3v via a 510R resistor and
\ the Cathode to PC-13.
\ This INVERTS the normal operation so LED is on at bootup.
\ PC-13 = HIGH, LED = OFF
\ PC-13 = LOW, LED = ON
\ ---------------------------------------------------------------------------\
 compiletoram
 \ compiletoflash

\ Configure PC-13 as Open Drain, OUTPUT.
\ PC13->output   \ See STM document RM008 page 172 0 1134
\ 23 22  21 20
\ CNF13  MODE13
\ 0  1   1  1
%11 20 lshift GPIOC_CRH bis!    \ GPIOC_CRH_MODE13 Output mode: 50MHz
%01 22 lshift GPIOC_CRH bis!    \ GPIOC_CRH_CNF13   Open Drain

: led.off   %1  13 lshift GPIOC_BSRR bis! ;
: led.on  %1 13 lshift GPIOC_BRR bis! ;

: blink
     10 0 do           \ blink 10 times, 1/2 second on, 1/2 second off.
        led.on
        500 ms.delay\ use ms.delay Word included in STM32F103C8-DIAGNOSTICS.bin
        led.off
        500 ms.delay
     loop
;

blink
```

Fun Stuff

Interactivity

Now you have your blinky working, try this:

```
led.on
```

The LED should turn on ?

```
led.off
```

The LED should have turned off ?

This is the *INTERACTIVE* nature of Forth, instant real world feedback. No need to compile and flash as is so common with programming languages like C, because with Forth you can obtain instant hardware feedback from the terminal.

Register Pretty Printing

Now try this:

```
gpioc.
```

Note

the period “.” at the end of the command

You should have a list of the *CURRENT STATE* of EVERY GPIOC REGISTER displayed ?

The idea is to view this in conjunction with the STM32F103 ST Micro Reference Manual RM0008 Rev 20 open on page 171/1134.

GPIOC_ODR

Enter this into the terminal:

```
led.on
```

The GPIOC OUTPUT DATA REGISTER (ODR) is the one which is telling the LED to light via BIT 13. To view it, enter:

```
GPIOC_ODR.
```

Note

the period “.” at the end of the command

It should look like this:

```
gpioc_odr.
GPIOC_ODR  $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
```

No bits are HIGH, the LED is ON because BIT 13 is LOW, due to the Open Drain wiring.

Now enter:

```
led.off
```

And again enter :

```
GPIOC_ODR.
```

See what has changed ? BIT 13 is now HIGH.

```
gpioc_odr.
GPIOC_ODR  $00002000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0
```

Real Time Views

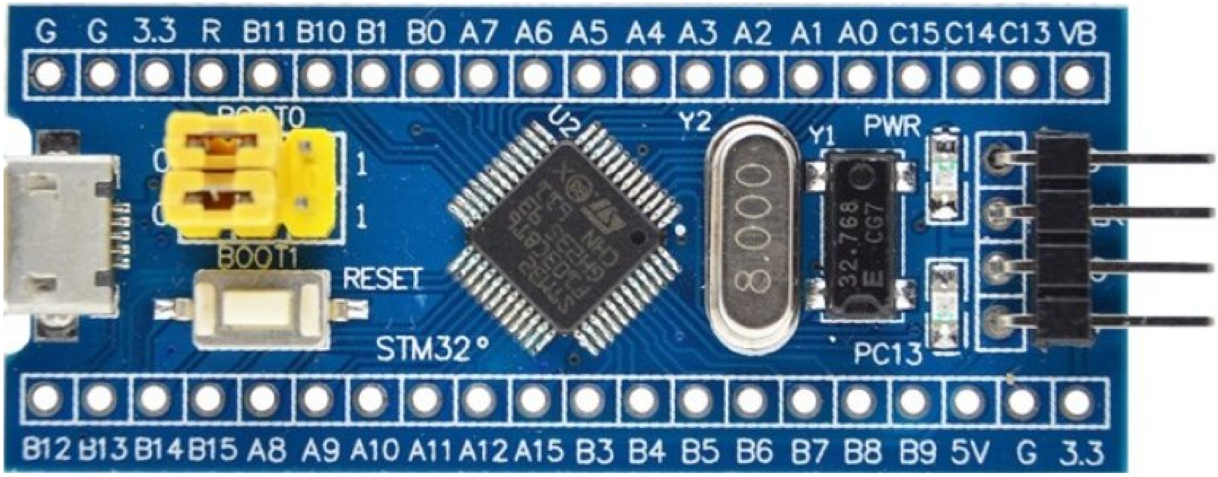
You can view IN REAL TIME the contents of all registers that have support Words loaded into your chip.

To view all the support Words in your chip right now, type the following and you will see that there are about 760 support programs (Words) in your chip ready to be used !

```
Words4
```

The STM32F103C8 DIAGNOSTICS image only has a small subset of programs because it had to be smaller than 64kB to fit any Blue Pill chip, but you can get and even make images with EVERY STM32F103 register Word ready to use.

Blue Pill Starter Kit (STM32F103)


G G 3.3 R B11 B10 B1 B0 A7 A6 A5 A4 A3 A2 A1 A0 C15 C14 C13 VB
BOOT0
BOOT1
RESET
STM32
Y2
Y1
PWR
PC13
8.000
B12 B13 B14 B15 A8 A9 A10 A11 A12 A15 B3 B4 B5 B6 B7 B8 B9 5V G 3.3

The Blue Pill board is very popular due to the very low cost and many new Forth users who are not familiar with the MCU used in this board find that it’s a major task to even get a LED on the board to turn on/off.

The MCU in the Blue Pill is quite old and lacks the capabilities of more modern STM chips, however it is still extremely complex and the new user will soon be researching a 1096 page **`Technical Reference PDF`** to understand how everything works.

Do have a quick look at my anti Blue Pill rant *especially if you have not yet bought one* ?

This Starter Kit contains files to upload into your Blue Pill and a tutorial to help make mastering the STM32F103 MCU in general, and the Blue Pill LED in particular a bit easier. It covers basic RCC and GPIO usage. Everything is freely available under Open Source Licenses.

You only need a serial terminal and file editor on your PC, no GCC, no GDB, no Eclipse, no massive GigaByte software downloads. This will run on **any OS** in a serial terminal.

With Forth on a Blue Pill, a complete INTERACTIVE development system, including the Starter Kit files is built into the MCU using only 26.5 KB of Flash.

Note

The methods shown here are just ONE WAY to do this. Forth can do this any way YOU want!

See also

STM32F103C8 V/S GD32VF103Cx

Prerequisites

Terminal

Connect a 3.3v/usb dongle to USART1 pins PA9 (TX)—> Dongle (RX) PA10 (RX) —> Dongle (TX)

1) Mecrisp-Stellaris must be loaded onto your Blue Pill and running

So when you reboot the board you get a message similar to this on the terminal.

```
Mecrisp-Stellaris RA 2.5.1 for STM32F103 by Matthias Koch
```

Download Mecrisp-Stellaris RA 2.5.1

You can DOWNLOAD the same self booting Mecrisp-Stellaris binary file I use for this tutorial `HERE`.

Problems Flashing

If you need help flashing Mecrisp-Stellaris onto your Blue Pill, see here.

What is all this about anyway ?

If you have just stumbled across this page and have no idea what any of this is about, this link may help.

2) You must be able to upload Forth files via your terminal

Without experiencing the infamous, and easily fixed “lockup”. see Common Problems Page

3) You *should* have the STM32F103xx Reference Manual PDF open

The manual I’m referring to is Doc ID 13902 Rev 14 dated October 2011, DOWNLOAD it HERE: `Technical Reference PDF`, yes it’s 1096 pages.

The Kit

The kit contains 7 files:

File	Description	Download Link

File	**Description**	**Download Link**
Technical Reference	STM Doc ID 13902 Rev 14 dated October 2011	**`Technical Reference PDF`**
1b.fs	Print support file, must be loaded first	**`1b.fs`**
memmap.fs	Memory map file, must be loaded second	**`memmap.fs`**
bitfields.fs	Configuration templates file, not to be loaded. copy and paste selections	**`bitfields.fs`**
bp-memstat.fs	This Word tells you how much Flash and Ram you have used	**`bp-memstat.fs`**

File	Description	Download Link
words4.fs	Lists all the Words (programs, subroutines, functions) on your Blue Pill	**`words4.fs`**
bluepill.support.fs	Words used in this kit	**`bluepill.support.fs`**

Using the Kit

Load the five files listed below:

```
1b.fs
memmap.fs
bp-memstat.fs
words4.fs
bluepill.support.fs
```

into your Blue Pill via the Mecrisp-Stellaris terminal **in the exact sequence shown here top to bottom**. These should load into Flash without any errors and finish with a “ok” at the cursor.

Note

If you mess up here, perhaps load everything twice and start getting “Redefine” Warnings (this isn’t a ERROR) you can easily delete everything you have loaded (but not Mecrisp-Stellaris) with the command “eraseflash” and start again.

Utilities

First, a couple of useful utility files, bp-memstats.fs which shows how much Flash and RAM you have and words4.fs which shows all the Forth programs loaded (and available). Both of these are now loaded into your Blue Pill.

Memstats

All the doc I’ve read claims the Blue Pill chip has 20KB of RAM, so that’s what the Word “free” assumes it has.

The actual amount of Flash on your chip is stated in a register, so “free” looks that up. My chip (not a Blue Pill) only has 65KB Flash. Your results may be different as I read that many Blue Pills have 128KB Flash MCU’s.

My LC Mini

```
free

free (bytes)
FLASH.. TOTAL REPORTED: 65536 USED: 26460 FREE: 39076
RAM.... TOTAL PRESET: 20000 USED: 628 FREE: 19372
```

Note

Mecrisp-Stellaris, and all the Starter Kit files are loaded into Flash so the small amount of RAM shown used is for Mecrisp-Stellaris.

This shows that the ENTIRE Mecrisp-Stellaris **INTERACTIVE Forth program** plus the five programs of the Starter Kit **use only 26.5 KiloBytes on your chip** leaving at least

39KB of Flash (for a STM43F103 with 64KB total Flash) and 19.3KB of RAM available for user programs.

In the **EMBEDDED Forth** world, 39KB of Flash is a MASSIVE amount of memory. If your Blue Pill has 128KB of Flash, then you would have 103 KB of Flash free!

Mecrisp-Stellaris Forth is the SMALLEST complete INTERACTIVE EMBEDDED PROGRAMMING LANGUAGE for STM32 that is available under the GPL to my knowledge.

A Blue Pill With a NON STM32F103 MCU ?

Thanks to ‘Michael Jeffery’ for this feedback. He has a STM32F103 clone, the CKS CS32F103

```
free (bytes)
FLASH.. TOTAL REPORTED: 131072 USED: 42988 FREE: 88084
RAM.... TOTAL PRESET: 20000 USED: 1072 FREE: 18928
```

Words4

This utility displays the Words (also known as programs, subroutines or functions in other programming languages) currently loaded and available on your Blue Pill.

```
words4

--- Mecrisp-Stellaris RA 2.5.1 ---
2dup                2drop               2swap
2nip
2over               2tuck               2rot
2-rot
2>r                 2r>                 2r@
2rdrop
d2/                 d2*                 dshr
dshl
dabs                dnegate             d-
d+
s>d                 um*                 m*
ud*
udm*                */                  */mod
u*/

... lots more Words ...

irq-tim7            irq-usbfs           --- Flash Dictionary
---
b32loop.            b32sloop.           1b.
bin.
RCC                 RCC_CR              RCC_CFGR
RCC_CIR
RCC_APB2RSTR        RCC_APB1RSTR        RCC_AHBENR
RCC_APB2ENR
RCC_APB1ENR         RCC_BDCR            RCC_CSR
RCC_CR.
RCC_CFGR.           RCC_CIR.            RCC_APB2RSTR.
RCC_APB1RSTR.
RCC_AHBENR.         RCC_APB2ENR.        RCC_APB1ENR.
RCC_BDCR.
RCC_CSR.            RCC.                GPIOA
GPIOA_CRL
GPIOA_CRH           GPIOA_IDR           GPIOA_ODR
GPIOA_BSRR
GPIOA_BRR           GPIOA_LCKR          GPIOA_CRL.
GPIOA_CRH.
GPIOA_IDR.          GPIOA_ODR.          GPIOA_BSRR.
GPIOA_BRR.
GPIOA_LCKR.         GPIOA.              GPIOB
GPIOB_CRL
GPIOB_CRH           GPIOB_IDR           GPIOB_ODR
GPIOB_BSRR
GPIOB_BRR           GPIOB_LCKR          GPIOB_CRL.
GPIOB_CRH.
GPIOB_IDR.          GPIOB_ODR.          GPIOB_BSRR.
GPIOB_BRR.
GPIOB_LCKR.         GPIOB.              GPIOC
GPIOC_CRL
GPIOC_CRH           GPIOC_IDR           GPIOC_ODR
GPIOC_BSRR
GPIOC_BRR           GPIOC_LCKR          GPIOC_CRL.
GPIOC_CRH.
GPIOC_IDR.          GPIOC_ODR.          GPIOC_BSRR.
GPIOC_BRR.
```

```
GPIOC_LCKR.         GPIOC.              RCC_APB2ENR_ADC1EN
RCC_APB2ENR_ADC1-DIS
ADC1EN?             GPIOC_CRH_MODE13    GPIOC_CRH_MODE13-
clear
GPIOC_CRH_CNF13     GPIOC_CRH_CNF13-clear
GPIOC_BSRR_BS13     GPIOC_BSRR_BR13     GPIOC_13_clear
GPIOC_13_output
pc13.on             pc13.off            ledon
ledoff
flashfree           ramfree             free
words4
 ok.
```

Go ahead and type in “free” or “words4” into your terminal (don’t forget to hit the ENTER key after each Word), have a play with them, get comfortable with using them.

Note

“free” and “words4” are just TWO of many, many diagnostic and development Words available to Mecrisp-Stellaris Forth users.

STM32F103 Basics

Power up/reset

At power-up and reset the STM32F103 has **all the internal Peripherals switched off** and normally only the internal RC clock is running at this time. However if you have Mecrisp-Stellaris running on your Blue Pill, all the important configuration *HAS ALREADY BEEN DONE* and vital Peripherals are already ENABLED and configured for you.

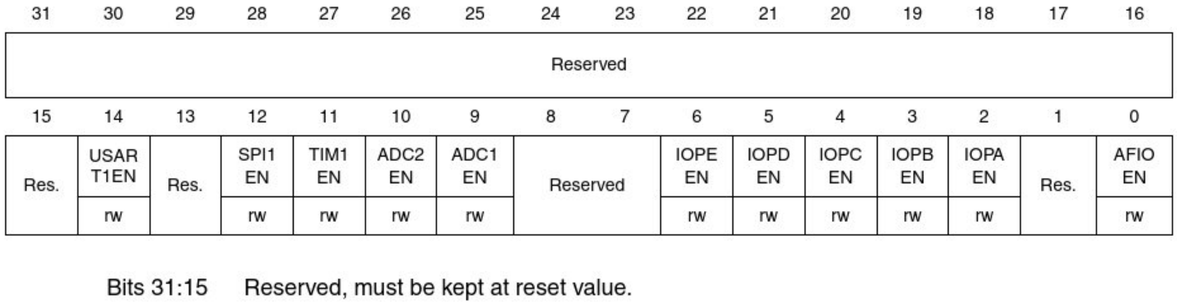
RCC

The STM32F has one **VERY IMPORTANT REGISTER GROUP** named **“RCC”** which stands for “Reset and Clock Control”. **The RCC controls all the Peripherals, whether they are ON OR OFF** plus other important settings.

Which Peripherals are ENABLED on your Blue Pill ?

1. Turn to page 142 of the **`Technical Reference PDF`** and you will see the following table.

APB2 PERIPHERAL CLOCK ENABLE REGISTER (RCC_APB2ENR)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

Bits 31:15 Reserved, must be kept at reset value.

2) now type the following command into your terminal:

```
RCC_APB2ENR.
```

Note

the “.” period at the end of the Register name, this is important.

And you will get the following Register Pretty Print which shows the values of the “Bitfields” in the RCC_APB2ENR register on your Blue Pill *RIGHT NOW*, in real time.

```
RCC_APB2ENR.
RCC_APB2ENR (read-write) $0000403D
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0
1
ok.
```

Analysis

Comparing the picture and the print on your screen we can see that all the GPIO ports and USART1 have been ENABLED already by Mecrisp-Stellaris as part of its initilization routine.

Bit	Enabled?	Mnemonic	Description
0	Y	AFIOEN	Alternate Functions Enable
2	Y	IOPAEN	GPIOA ENABLE
3	Y	IOPBEN	GPIOB ENABLE
4	Y	IOPCEN	GPIOC ENABLE
5	Y	IOPDEN	GPIOD ENABLE
9	N	ADC1EN	Analog to Digital Converter 1 ENABLE

Bit	Enabled?	Mnemonic	Description
14	Y	USART1EN	USART1 ENABLE

Self Test

Is TIM1 ENABLED ?

ADC1

ADC1 is NOT enabled, how can we enable ADC1 to use it ?

Bitfields.fs

- File bitfields.fs was automatically generated to make configuring Bitfields easier than having to read the Datasheets and work out every single thing.
- bitfields.fs is a TEMPLATE collection, the Word names are automatically generated and usually changed by the programmer (you) to suit the purpose at hand.
- Each template in bitfields.fs is designed to SET a BIT or BITS using the “BIS!” Word. This is changed if RESETTING (BIC!) or TESTING (BIT@) is required instead.
- Because the contents of bitfields.fs are automatically generated from CMSIS-SVD data they make use of the “LSHIFT” command which SHIFTS BITS LEFT into their CMSIS-SVD specified bit positions.

Note

Check the Dictionary for what BIS!, lshift, BIC! and BIT@ actually do.

Open bitfields.fs in your editor and search for “ADC1EN”. This spelling is ARM CMSIS-SVD compliant. Your search will find this line:

```
: RCC_APB2ENR_ADC1EN   %1 9 lshift RCC_APB2ENR bis! ;  \
RCC_APB2ENR_ADC1EN    ADC 1 interface clock  enable
```

The name of the Word is “RCC_APB2ENR_ADC1EN” which is simply formed from the ARM CMSIS-SVD FILE for this chip. As written, this Word when executed on the MCU will set BIT 9 of the RCC_APB2ENR Register, which will ENABLE ADC1. Don’t enter anything, RCC_APB2ENR_ADC1EN has been previously loaded with “bluepill.support.fs”.

Enabling ADC1

Now execute RCC_APB2ENR_ADC1EN

```
RCC_APB2ENR_ADC1EN  ok.
```

Examine RCC_APB2ENR again and BIT 9 is now SET, ENABLING ADC1.

```
RCC_APB2ENR.
RCC_APB2ENR (read-write) $0000423D
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0
1
ok.
```

Disabling ADC1

But this Starter Kit doesn’t have ADC1 support so it should be disabled now. To get support for all the Peripherals **`Download: SVD2FORTH-V3-STM32F103C8T6.zip release`** WHEN you have worked thru to the end of this Starter Kit. NOTE: it needs Unix, Linux or FreeBSD to manipulate the files.

To disable ADC1 a new Word is needed, it’s a variation on RCC_APB2ENR_ADC1EN above with a name change and using BIC! instead of BIS! Don’t enter anything, RCC_APB2ENR_ADC1-DIS has been previously loaded with “bluepill.support.fs”.

```
: RCC_APB2ENR_ADC1-DIS   %1 9 lshift RCC_APB2ENR bic! ;
```

Run RCC_APB2ENR_ADC1-DIS

```
RCC_APB2ENR_ADC1-DIS  ok.
```

Read RCC_APB2ENR, NOTE: BIT 9 is now RESET, DISABLING ADC1

```
RCC_APB2ENR.
RCC_APB2ENR (read-write) $0000403D
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0
1
ok.
```

Those paying attention would have noticed that although we changed BIT 9 **NONE OF THE OTHER BITS BITS IN RCC_APB2ENR WERE AFFECTED**

Test ADC1 STATE ?

A Word can be made to TEST the state of the ADC1EN BIT, so that looking up the values of RCC_APB2ENR and searching for BIT 9 isn’t needed, this is how that Word could be designed. Don’t enter anything, ADC1EN? has been previously loaded with “bluepill.support.fs”.

```
: ADC1EN?   %1 9 lshift RCC_APB2ENR bit@
     if ." ADC1 is ENABLED "
        else ." ADC1 is DISABLED "
     then cr
;
```

Now run ADC1EN?

```
ADC1EN? ADC1 is DISABLED
```

Now ENABLE ADC1

```
RCC_APB2ENR_ADC1EN  ok.
```

Rerun ADC1EN?

```
ADC1EN? ADC1 is ENABLED
ok.
```

Conclusion

Bitfields.fs contains Words with names auto-generated from the CMSIS-SVD which can be searched using names from the Technical Reference. Their Word names can be changed to reflect your usage and they can set BITS by default or be altered to RESET or even TEST bits as shown above.

Lots More Registers in RCC!

Q: But there are more Registers in the RCC, do I have to type the name for each one to see it ?

A: No, type the Group Name, ie “RCC” (don’t forget the trailing period)

```
rcc.

rcc.
RCC_CR () $00005083
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1
1

RCC_CFGR () $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC_CIR () $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC_APB2RSTR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC_APB1RSTR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC_AHBENR (read-write) $00000014
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
0
```

```
RCC_APB2ENR (read-write) $0000403D
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0
1

RCC_APB1ENR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC_BDCR () $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

RCC_CSR () $1C000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
ok.
```

GPIO

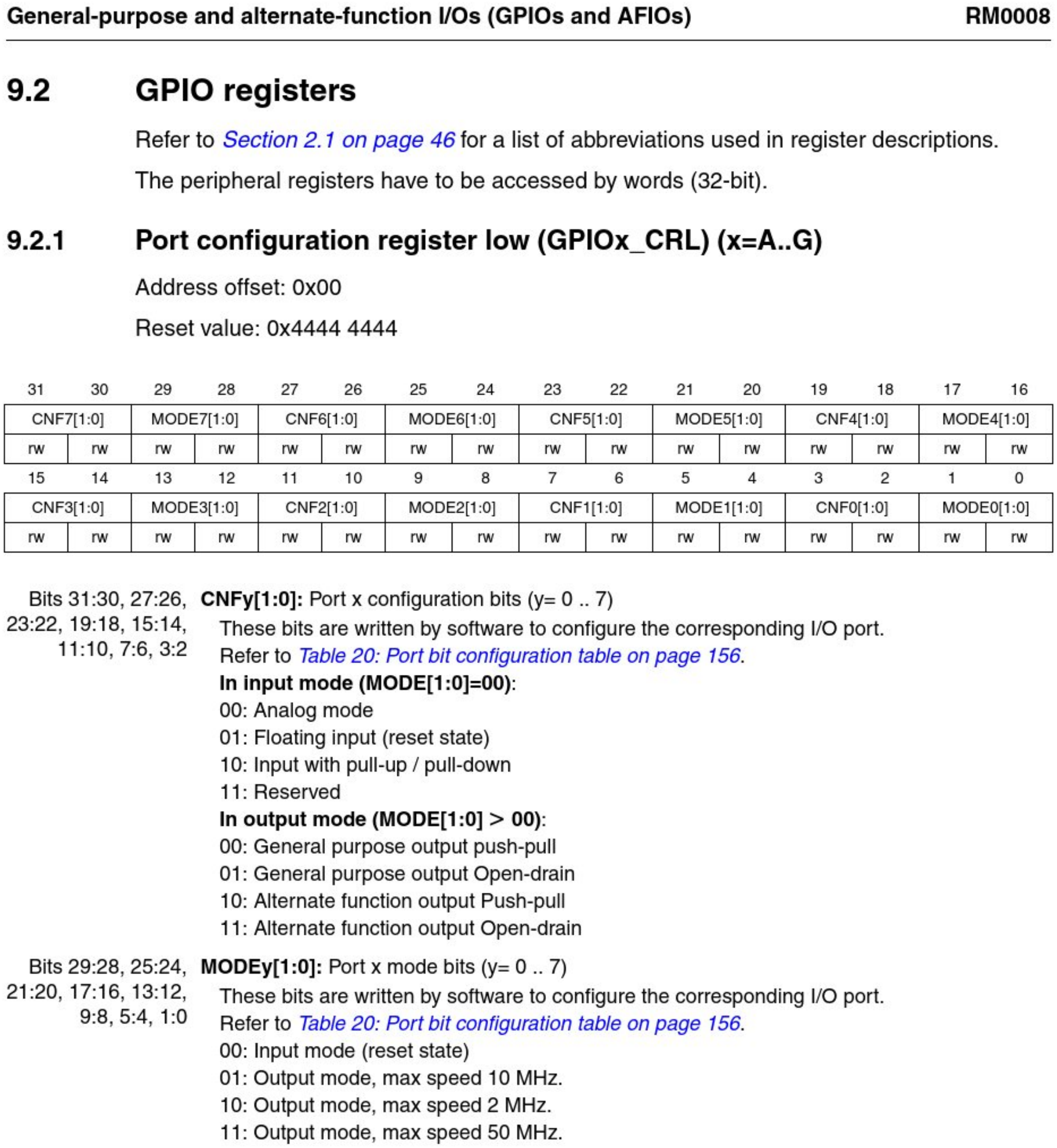
Now onto the STM3F103 GPIO’s, this is the most confusing part of this tutorial and the age of this chip shows here in the unwieldy way the GPIO’s are configured. This is much simpler in later series such as Cortex-M0.

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain
- Output push-pull
- Alternate function push-pull
- Alternate function open-drain

Turn to page 166 of the `Technical Reference PDF` and you will see the following table.

Port configuration register low (GPIOx_CRL)



General-purpose and alternate-function I/Os (GPIOs and AFIOs) **RM0008**

9.2 GPIO registers

Refer to *Section 2.1 on page 46* for a list of abbreviations used in register descriptions.

The peripheral registers have to be accessed by words (32-bit).

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 156*.

In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 156*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

Examine GPIOA

Now we know that the GPIO’s are ENABLED we can see the data in the GPIOA Peripheral the same way you saw the RCC data earlier.

GPIOA-CRL

As seen from the GPIOx_CRL table above, GPIOA_CRL controls the configuration of GPIOA bits 0 to 7. If you think that this is confusing, you’re right. Remember *old chip* Yet another reason *not* to use the Blue Pill.

```
GPIOA_CRL.
GPIOA_CRL (read-write) $44444444
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0
```

Note that there are eight GPIO’s here, GPIOA 0 to 7 and are all the same value as specified in groups of 4 bits, namely %0100 or HEX 0x4.

This means that GPIOA 0 to 7 are in their default value (after reset) which hasn’t been changed.

Decoding GPIO Config

- First find the value in the MODE0 Bitfield, this will determine if the BIT is INPUT or OUTPUT.
- Then look in CNF0 table either under INPUT or OUTPUT depending on the MODE0 value.

GPIOA_CRH

As seen from the table above, GPIOA_CRH controls the configuration of GPIOA BITS 7 to 15

```
GPIOA_CRH.
GPIOA_CRH (read-write) $888448A4
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0
0
```

Note that most of the GPIO’s are in the RESET DEFAULT configuration but with a few exceptions, namely BITS 8,9,12, 13 and 14.

BIT	VALUE	Mode
9	0xA	OUTPUT 2 MHz, ALTERNATE FUNCTION, PUSH-PULL
10,13,14,15	0x8	OUTPUT 2 MHz, PUSH-PULL

Self Test

Whats so special about GPIOA-9 ?

Print ALL GPIOA Contents

This is done in the same way as the RCC. above, only this time the Register Group is named “GPIOA”

```
gpioa.
GPIOA_CRL (read-write) $44444444
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0

GPIOA_CRH (read-write) $888448A4
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0
0

GPIOA_IDR (read-only) $0000BD7F
```

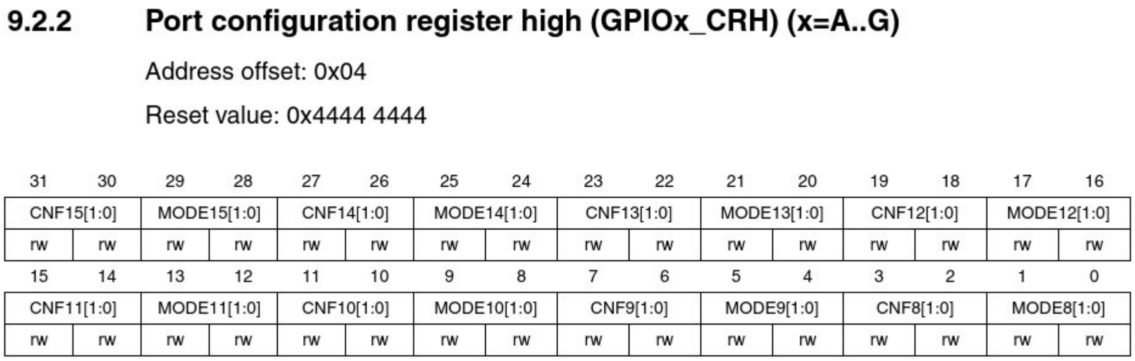
```
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1
1

GPIOA_ODR (read-write) $0000A400
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0

GPIOA_BSRR (write-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_BRR (write-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_LCKR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
ok.
```

Blue Pill LED

The Blue Pill User LED is located at GPIOC-13 so in order to turn it ON it we will need to configure PC-13 as a OUTPUT first.

Check GPIOC-13 configuration

GPIOC-13 is controlled by GPIOC_CRH.



9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Where is BIT 13 ?

Look in the Reference Manual PDF, page 167 for “Port configuration register high (GPIOx_CRH)” and you will see that

- MODE13[1:0] are BITS 20 and 21
- CNF13[1:0] are BITS 22 and 23.

As before, we find that all the BITS are in the RESET DEFAULT of INPUT, FLOATING, including PC13.

```
GPIOC_CRH.

GPIOC_CRH.
GPIOC_CRH (read-write) $44444444
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0
ok.
```

Open bitfields.fs in your editor and search for “GPIOC_CRH” then look for “13” and you should see these lines, which control the Bitfields for MODE13 and CNF13. Just take note of these Words, don’t enter or run them. They are for your information only and have been previously loaded with “bluepill.support.fs”

```
: GPIOC_CRH_MODE13   ( %XX -- ) 20 lshift GPIOC_CRH bis! ;  \
GPIOC_CRH_MODE13   Port n.13 mode bits
: GPIOC_CRH_CNF13    ( %XX -- ) 22 lshift GPIOC_CRH bis! ;  \
GPIOC_CRH_CNF13    Port n.13 configuration  bits
```

These templates will be modified to configure PC-13 as a OUTPUT.

Firstly don’t assume MODE13[1:0] and CNF13[1:0] are in the default modes, we need to make this work under any pre-existing configurations.

Note

This solution is one of a infinitely variable set of solutions, it’s not the best or the most efficient.

PC-13 Output Word

Just take note of these Words, don’t enter or run them. They are for your information only and have been previously loaded with “bluepill.support.fs”

```
: GPIOC_13_clear ( -- )         \ Clear all GPIOC_13 config
bits
  %11 20 lshift GPIOC_CRH bic!  \ GPIOC_CRH_MODE13  0,0
  %11 22 lshift GPIOC_CRH bic!  \ GPIOC_CRH_CNF13   0,0
;

: GPIOC_13_output ( -- )        \ Output, Push-Pull:
MODE13:1,0   CNF13:0,0
  GPIOC_13_clear                \ MODE13:0.0  CNF13:0,0
  %10 20 lshift GPIOC_CRH bis!  \ MODE13:1,0
;
```

PC-13 Control Words

Just take note of these Words, don’t enter or run them. They are for your information only and have been previously loaded with “bluepill.support.fs”.

```
: pc13.high ( -- ) GPIOC_BSRR_BS13 ;
: pc13.low  ( -- ) GPIOC_BSRR_BR13 ;

: GPIOC_13_output ( -- )        \ Output, Push-Pull:
MODE13:1,0   CNF13:0,0
   GPIOC_13_clear               \ MODE13:0.0  CNF13:0,0
   %10 20 lshift GPIOC_CRH bis! \ MODE13:1,0  SET PC13 as
PUSH-PULL
   pc13.high                    \ Start with LED off
;

: ledon pc13.low ;   \ The Blue Pill obviously has the LED
anode connected to +3.3v and the Cathode to PC13
                     \ (plus a resistor somewhere). When PC13
is HIGH the LED is OFF.
                     \ This means PC13 could be in "open
collector mode" instead of "push-pull"and work fine also.
: ledoff pc13.high ;
```

LED On and Off

This Word configures PC-13 as a output, so type it into your terminal and hit enter.

```
GPIOC_13_output
```

Now type the following Words into your terminal, one at a time and hit enter. The LED should turn ON or OFF as instructed.

```
ledon

ledoff
```

The End

Congratulations, you have finished the Blue Pill Starter Kit !

Exercise for the reader

GPIOC_13_output is used to configure GPIOC_13 as a “PUSH-PULL” output, however this is the wrong MODE to use because the Blue Pill wiring expects a “OPEN-DRAIN”, can you change the code in “bluepill.support.fs” so that Open Drain is used instead ?

Don’t forget to verify that OPEN_DRAIN is actually in use by reading the GPIOC_CRH Register.

See some user contributed EXAMPLE CODE ?

What’s Next ?

That’s a BIG question!

If you want to keep using the Blue Pill then that 1096 page Technical Reference is waiting to be read, allow at least 5 years to become somewhat conversant with this 12 year old chip!

Perhaps you may like to consider other more modern STM32 chips instead, available as brand new STM Discovery or Nucleo boards for around $20 with integrated debugger/programmer ?

Chip	Speed MHz	On Board Peripherals	Comments

Chip	Speed MHz	On Board Peripherals	Comments
STM32L072	32	45	Low power Cortex-M0+
STM32F777	200	93	FAST and comprehensive

If you want to continue with Forth you could look at the Next Steps page ?

Starter Kit Testing

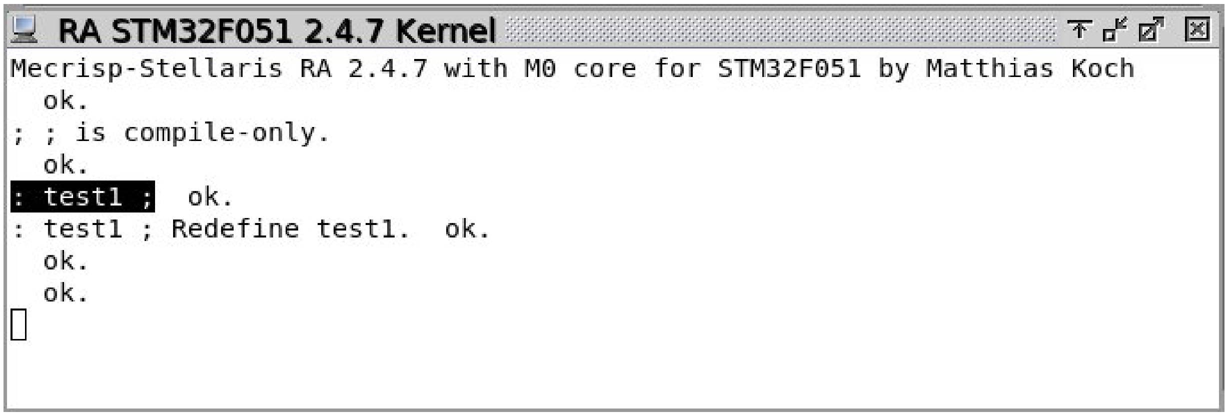
All Starter Kit Words and the LED were tested on a LC Mini which has the same STM32F103 chip as the Blue Pill.

Colourized Error Messages

One of the really fantastic things about Mecrisp-Stellaris is that one can easily hack it to suit ones requirements because the source code is GPL’d and Open, and the Author Matthias Koch is friendly and helpful.

Mecrisp-Stellaris has a on-chip compiler, which also emits ERROR MESSAGES as it finds them in Source Code that is being uploaded.

The errors are displayed in the Terminal screen as they occur, mixed in with the source code, and all displayed in the same default Terminal text colour.



```
Mecrisp-Stellaris RA 2.4.7 with M0 core for STM32F051 by Matthias Koch
  ok.
; ; is compile-only.
  ok.
: test1 ;  ok.
: test1 ; Redefine test1.  ok.
  ok.
  ok.
```

Some clever Serial Terminals such as E4THCOM parse for the known error messages recolouring them and stopping the upload. E4thcom is distributed as a binary and only runs on Linux.

In other ARTICLES I describe the error response system I deploy, namely :

1. Lighting a bright BLUE led on the Target when a ERROR is detected

2. Stop the upload at the actual error via Hardware Handshaking

Note

Both these methods tap into the On Chip Compiler Error Message system. It knows all about errors, why not use it ?

Benefit to New Forth Users

When I started learning Forth, I’d often miss a small error message as the upload streamed by in the Terminal. The small error would often lead to subsquent errors that I’d chase not knowing the true cause. It’s my hope that new Forthers may also find this method a time saver once implemented.

YouTube Video

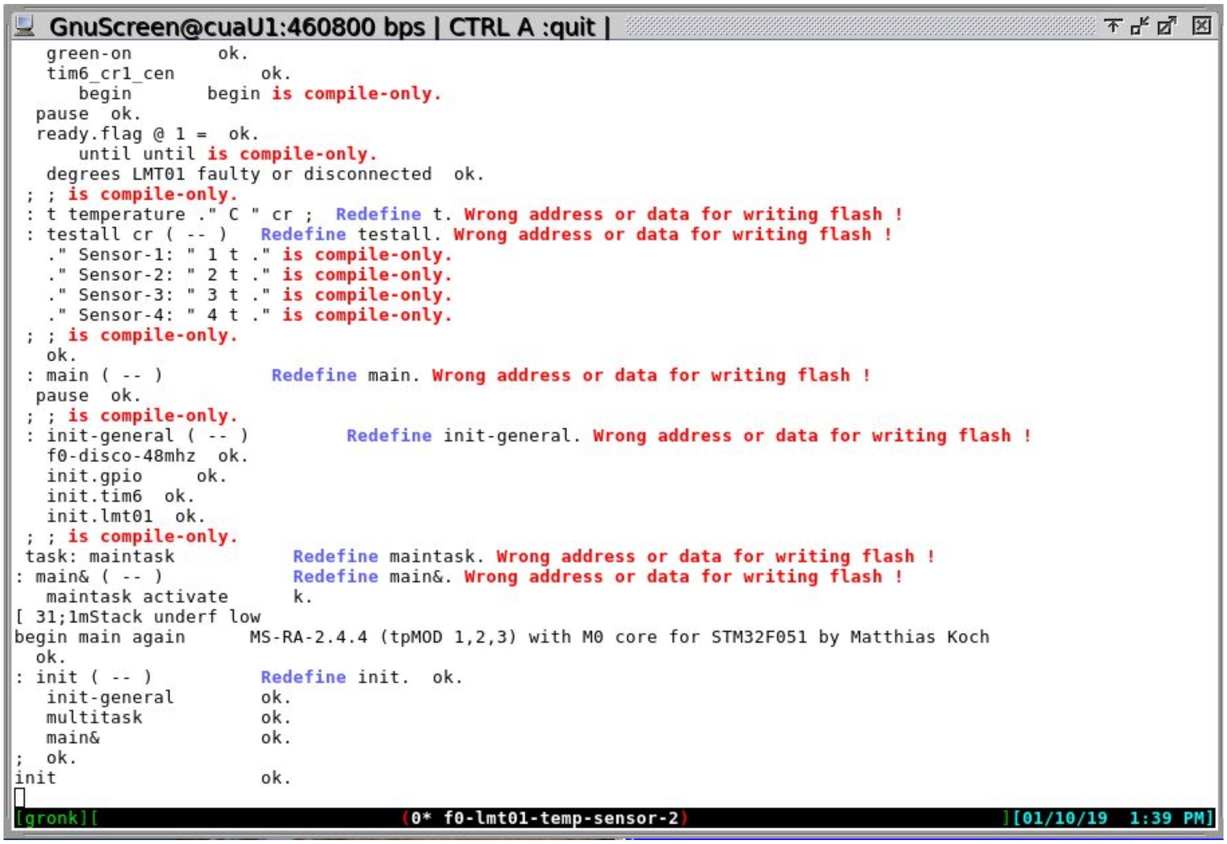
https://youtu.be/vAGTKLZq0PM

Adding Colourized Error Messages to Mecrisp=Stellaris

In this article I describe a somewhat easier to deploy method of error message colouring which should work across the entire Mecrisp-Stellaris MCU range with only a couple of Kernel Source code mods, and a Kernel REBUILD.

Colorized Error Reporting

This errors in this picture were contrived by compiling the source to Flash repeatedly until the Flash was full.



```
   green-on        ok.
   tim6_cr1_cen        ok.
      begin       begin is compile-only.
  pause  ok.
  ready.flag @ 1 =  ok.
      until until is compile-only.
   degrees LMT01 faulty or disconnected  ok.
 ; ; is compile-only.
 : t temperature ." C " cr ;  Redefine t. Wrong address or data for writing flash !
 : testall cr ( -- )   Redefine testall. Wrong address or data for writing flash !
   ." Sensor-1: " 1 t ." is compile-only.
   ." Sensor-2: " 2 t ." is compile-only.
   ." Sensor-3: " 3 t ." is compile-only.
   ." Sensor-4: " 4 t ." is compile-only.
 ; ; is compile-only.
   ok.
 : main ( -- )          Redefine main. Wrong address or data for writing flash !
  pause  ok.
 ; ; is compile-only.
 : init-general ( -- )         Redefine init-general. Wrong address or data for writing flash !
   f0-disco-48mhz  ok.
   init.gpio     ok.
   init.tim6  ok.
   init.lmt01  ok.
 ; ; is compile-only.
 task: maintask          Redefine maintask. Wrong address or data for writing flash !
: main& ( -- )           Redefine main&. Wrong address or data for writing flash !
   maintask activate      k.
[ 31;1mStack underf low
begin main again      MS-RA-2.4.4 (tpMOD 1,2,3) with M0 core for STM32F051 by Matthias Koch
  ok.
: init ( -- )          Redefine init.  ok.
   init-general        ok.
   multitask           ok.
   main&               ok.
;  ok.
init                   ok.

[gronk][                     (0* f0-lmt01-temp-sensor-2)                       ][01/10/19  1:39 PM]
```

CRITICAL: RED Colouring

Sourcefile: mecrisp-stellaris-X.X.X/mecrisp-stellaris-source/common/datastackandmacros.s

Note

This adds the ANSI escape sequence for the color RED indicating CRITICAL errors. Lines which start with “-” need to be replaced with lines starting with “+ ”

```
.macro Fehler_Quit_n Meldung
bl dotgaensefuesschen
.byte 8f - 7f         @ Compute length of name field.


-   7:      .ascii "\Meldung\r\n"
```

```
+  7:      .ascii "\x1B[31;1m\Meldung\x1B[31;0m\r\n"

8:      .p2align 1
.endm
```

WARNING: BLUE Colouring

Sourcefile: mecrisp-stellaris-X.X.X/mecrisp-stellaris-source/common/compiler-flash.s

Note

This adds the ANSI escape sequence for the color BLUE indicating a WARNING

```
-    write "Redefine "
+    write "\x1B[34;1m Redefine\x1B[34;0m "
```

Now Rebuild the Kernel

REBUILD The Kernel

STM32F0xx Baudrate Calculator

This Program calculates baudrates by reading MCU configuration. Only works with USART1 16 bit oversampling which is the reboot default and will exit if 8 bit oversampling is in use.

It uses Mecrisp-Stellaris s31.32 fixed point support to calculate the Baudrate to two decimal places so you can choose the best BRR integer to use when setting up your terminal baudrate.

f0-baudrate-calculator.fs

```
$40021000 constant RCC ( Reset and clock control )
RCC $4 + constant RCC_CFGR ( Clock configuration register
RCC_CFGR )
$40013800 constant USART1 ( Universal synchronous
asynchronous receiver  transmitter )
USART1 $0 + constant USART1_CR1 ( Control register 1 )

8000000 constant clock  \ default, change if not.
: rcc_cfgr_pllmul? ( -- ) %1111  18  lshift $40021004 @ and
18 rshift ;  ( PLL Multiplication Factor )

: clock_multiplier? ( -- ) RCC_CFGR_PLLMUL? 2 + ;  \ clock
multiplier value

: usart1_cr1_over8? ( oversampling8? -- true )  %1 15 lshift
usart1_cr1 bit@ ;  \ oversampling mode. reboot default: 16

: oversampling? ( -- )
  usart1_cr1_over8?  if ." USART: 8 bit oversampling ! MUST
use 16 bit oversampling" exit then
;

: >s31.32 ( u -- s31.32 ) 0 swap ;       \ convert to a
```

```
s31.32 number

: brr? ( desired baud rate -- brr )       \ brr =  ((pll
multiplier * clock)/desired baud)/2
  >r
  clock_multiplier?  clock *  >s31.32    \ calculate bus
clock, convert to s31.32
  r@ >s31.32                             \ desired baud rate
  f/                                     \ f. divide
  2  >s31.32                             \ last step is
divide by 2
  f/                                     \  f. divide
  2                                      \ only display 2
comma places
  ." for a baud rate of " r> . ." the usart1_brr should be: "
f.n cr
;

: baud?  ( -- ) cr cr
  oversampling?
  ." CLOCK: " clock . ." PLL multiplier: " clock_multiplier?
. ." BUS CLOCK: " clock clock_multiplier? * . ." Hz " cr
  115200 brr?
  460800 brr?
  921600 brr?
  1843200 brr?
  cr
;
```

Program Output

From my STM32F051 Discovery Board.

```
CLOCK: 8000000 PLL multiplier: 6 BUS CLOCK: 48000000 Hz
for a baud rate of 115200 the usart1_brr should be: 208,33
for a baud rate of 460800 the usart1_brr should be: 52,08
for a baud rate of 921600 the usart1_brr should be: 26,04
for a baud rate of 1843200 the usart1_brr should be: 13,02
```

Gnu Screen

This page describes why I now use Gnu Screen with Forth, and why I believe it is the best choice in a modern Forth based development environment.

Historical Forth Development

Historically, Forth used a serial terminal, usually running at 9600 Baud with a file upload facility, such as Picocom. Forth interactivity and 9600 Baud uploads with only Mk1 Eyeball code error checking was fine back in the day, but that day was 30 years ago.

Modern Development Systems

Nowadays high level languages such as C running via CGB via OpenOCD or SWD to the target chip, are far more attractive than the historical Forth development, even if some of them such as the free STM owned Attolic Sudio require a quad core PC with 64GB ram for a responsive user experience.

Maths Library

Feb 2018, Andrew Palm, a mathematician, has created a square root, trig, and inverse trig functions for s31.32 fixed point numbers library **`fixpt-math-lib.fs`** fixpt-math-lib.fs.

See also

s31.32 Insights and A sunrise and sunset calculator

Millis

Millis times any event (like a stopwatch) using the Cortex-M Systick Counter with a Interrupt. Resolution is 0.1 milliseconds. Range is 0.1 ms to 71582 minutes.

This should run on any Cortex-M but check your constants for stk_csr,stk_rvr,stk_cvr and stk_calib have the correct memory addresses. The Systick values below are for a Cortex-M0.

This program was used to calibrate the Interactive Assembly and a accurate Cortex-M0 Delay Word projects.

f0-millis.fs

```
\ Program Name: f0-millis.fs
\ Date: Sat 12 Oct 2019 12:06:27 AEDT
\ Copyright 2019 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV2
\ This program does not require preloaded support files
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Chip: STM32F051, Board: STM32F0 Discovery Board
\ Clock: 8 Mhz using the internal STM32F051 RC clock
\ All register names are CMSIS-SVD compliant
\ Note: gpio a,b,c,d,e, and uart1 are enabled by Mecrisp-Stellaris Core.
\
\ This Program : Times a event using the Systick Counter and a Interrupt.
\ Resolution is to 0.1 milliseconds. Range is 0.1 ms to 71582 minutes
\ Chip is Cortex-M0 but should work on any Cortex-M


\ 8080 STK_RVR !    \ 1ms @ 8mhz, 808 = 0.1 mS @ 8Mhz

\ define systick memory mapping as it's not included in
```

```
cmsis-svd.
\ Should work on all Cortex-M chips, check values for non
Cortex-M0 chips.
$e000e010 constant stk_csr            \ systick control and
status register. r/w reset value = $00000000
$e000e014 constant stk_rvr            \ systick reload value
register.
$e000e018 constant stk_cvr            \ systick current value
register. r/w value unknown
$e000e01c constant stk_calib          \ systick calibration
value register. read only, $40001770 for the stm32f0

0 variable ms_counter                 \ 32 bits or ->
$ffffffff u. =  4294967295 ms, 4294967 seconds or 71582
minutes
: tickint                             \ tickint: sysTick
exception request enable
  %010 stk_csr bis!                   \ 1 = Counting down to
zero asserts the SysTick exception request.
;

: systick-handler
 1 ms_counter +!
;

: init-systick
  808 stk_rvr !                       \ systick calib for
0.1ms @ 8mhz clock
  %101 stk_csr bis!                   \ systick enable
  ['] systick-handler irq-systick !   \ 'hooks' the systick-
handler word (above) to the systick irq
  tickint
;

: pretty.print ( -- )                 \ insert one decimal
point
 0 <# # 46 hold #s #> type
;

: millis.reset
  0 ms_counter !                      \ reset the ms_counter
to zero
;

: millis.read                         \ read elapsed millis
```

```
since millis.reset
  ms_counter @                        \ this will be updated
by the systick-handler every systick
  pretty.print  ." ms " cr
;


\ -------------------------Example Usage--------------------
---------------- \

: do.stuff ( -- )                     \ Measure the time this
DO LOOP takes to complete
  1000000 0 do loop
;

: timeit cr
  init-systick
  millis.reset
  do.stuff
  millis.read
;

\ timeit
\ 653.6ms
\  ok.
```

What’s New ?

20 May 2020: Using Vim completion to easily insert bitfield configuration template Words into your Forth programs

16 May 2020: Bit Testing

07 May 2020: mecrisp-stellaris-2.5.3 PATCH to add CPU register printing, simple stack, no stair stepping, coloured error warnings, hardware handshaking, 460800 Baud. This is for Forthers developing on the Blue Pill etc and wishing to add some or all of these features to their kernel.

06 May 2020: Bitfields, Declare Early? This is an article about *EMBEDDED* development and maintenance coding style.

06 May 2020: Bitfields and Structures looking into methods used in Peripherals, Keil C v/s Svd2forth a comparison of how Keil C and Forth deal with embedded peripheral device configuration.

04 May 2020: Peripherals, Keil C v/s Svd2forth a comparison of how Keil C and Forth deal with embedded peripheral device configuration.

24 April 2020: STM32F103 BluePill ‘Developer’ image. It’s a bootable binary, requires a 128kB STM32F103C8 or STM32F103CB MCU and a 3.3v USB/serial dongle with CTS.

- https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/f103-developer-image.c9ebc3e9d6.README.html
- https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/f103-developer-image.tar.gz

17 April 2020: A short demo to show how to use Manfred Marhlow’s VOCs to define register identifiers.

25 March 2020: New page about MCU Documentation, where to find it, what you need.

23 March 2020: Updated STM32F1xx-Diagnostics to V1.4. Changed PB9 to permanent LOW for Maple Mini, menu is run from boot, Version number is automatically displayed in the menu. https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/stm32fxx-diagnostics-v1.4.bin

17 March 2020: Pictured Numerical Output; How to print without spaces ?

11 March 2020: STM32F0xx Touch Sensor (TSC) This project supplies a binary file ready to run on a STM32F0 Discovery Board. It provides a fully configured Touch Sensor and Menu so that various user created touch sensors may be tested and calibrated.

11 March 2020: Xon/Xoff flow control notes.

02 March 2020: Added causes for the Flash error message “Wrong address or data for writing flash !”

26 Feb 2020: New layout of the Interrupts Page, hopefully it makes a bit more sense now ?

23 Feb 2020: Updates to the general purpose interrupt driven systick timing library. There are now three options for elapsed time display:

1. Push milliseconds to TOS for computations
2. Print milliseconds
3. Print seconds, to three decimal places

23 Feb 2020: Updates to Benchmarking, added STM32F051 Cortex-M0 @ 75 MHz, and STM32F103 Cortex-M3 @ 75MHz

21 Feb 2020: Mecrisp Forth running as a CMSIS-RTOS thread on a STM32wb55. This is a fascinating development by Peter Schmid which allows the use of advanced and complex STM32WB peripherals such as USB, and BLE etc with tools such as CubeIDE, HAL and CubeMX, while also providing the development interactivity of Forth.

- http://spyr.ch/twiki/bin/view/Cosmac/ForthSTM32WB

20 Feb 2020: STM32F1xx-Diagnostics A new diagnostics which runs on Bluepills and should run on the Maple Mini board. Connect via USB. It is designed to determine if your chip has the ‘hidden’ 64kB Flash and test it, tell you if your chip is a fake STM32F103 (CS32F or a APM32) or even a ‘compatible’ GD32F103. See the link for more.

18 Feb 2020: Mecrisp-stellaris-2.5.3.tar.gz released which fixes a serious RA compiler comparison bug. New chip supported: STM32wb55 which has two Cortex-M cores, a M0 and a M4 and inbuilt Bluetooth™ 5.0 and IEEE 802.15.4 wireless ! Cudos to Peter Schmid for using CMSIS-SVD for his STM32wb55 support.

- https://sourceforge.net/projects/mecrisp/files/mecrisp-stellaris-2.5.3.tar.gz
- https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-wireless-mcus/stm32wb-series/stm32wbx5.html

06 Feb 2020 Exception Handling, how to DEBUG EXCEPTIONS and stop the very pesky *“Unhandled Interrupt 00000003 !”* message

05 Feb 2020: Led blinky to run on the STM32F103C8 DIAGNOSTICS for the Blue Pill Board.

24 Jan 2020: Rant Hackerspaces or SalesPlaces?

21 Jan 2020: Another rant, The Hard Way

16 Jan 2020: Updates to STM32F103C8 DIAGNOSTICS. Does your Blue Pill have a hidden EXTRA 64kB of Flash?

16 Jan 2020: Updates to GD32VF103 RISC-V, Mecrisp-Quintus Forth for the Longan Nano RISC-V Board

15 Jan 2020: Updates to my Blue Pill Rants page.

12 Jan 2020: Memory Stats refactored. Now a parameter driven general purpose memory stats program.

08 Jan 2020: General purpose interrupt driven systick timing library now takes the “1ms-cal-value” as a parameter and should work with any STM32Fxxxx.

08 Jan 2020: How to produce a bootable binary file from a Mecrisp-Stellaris project.

02 Jan 2020: Comparison: STM32F103C8 V/S GD32VF103Cx

29 Dec 2019: The Philosophy of Problems.

25 Dec 2019: Xmas Special! Youtube video: *Did your Blue Pill get a FREE 64kB Flash for Xmas ?* https://youtu.be/xpTvd1Zhe0U

25 Dec 2019: Forth YouTube Video List.

24 Dec 2019: Create does>, The Jewel of Forth. Examples of the all-important ‘Defining Words’ of Forth.

11 Dec 19: Does every STM32F103C8T6 (used in the Blue Pill board) report 64kB of Flash but actually have 128kB of Flash?, is this a “Public Secret” ?

09 Dec 19: BluePill 128kB bootable Forth USB image. Memstats, ChipID, most Peripheral registers, stack sensitive red-prompt, can switch live between USB and USART2 with hardware handshaking. Details in the tarball readme. https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/bluepill-128kb-usb.tar.gz/download

07 Dec 19: Buffers, how to use them.

23 Nov 19: A stack sensitive prompt

23 Nov 2019: Beginner advice

23 Nov 2019: Arrays, how to use them.

16 Nov 2019: Bit manipulation tips with examples.

16 Nov 2019: A link to “Blue Pill” Device ID’s collected using the Bootable Binary Image below.

14 Nov 2019: The Forth Outer Interpreter Design,(assembly language view) with syntax highlighting.

11 Nov 2019: DegreesF as well as degreesC; Rewrite and bug fixes of the ready to flash binary for a “Blue Pill Board” temperature sensor using the Texas Instruments LMT01. Also has bootable USB and ‘Device Electronic Signature’. 64KB of Flash required on the “Blue Pill”.

07 Nov 2019 Special 64KB Flash **USB** Blue Pill Image that can read the internal ‘Device Electronic Signature’ plus other utilities. Use it to check for *FAKE STM32F103 MCU’s* and let me know if it is useful ?

30 Oct 2019: Added s31.32 Insights by Andrew Palm. Very helpful if you are trying to get a grip on Fixed Point math.

27 Oct 2019: A one minute YouTube video showing the building of the 128KB STM32F103 developers image with RTS/CTS hardware handshaking . Basically it shows the 2000 byte source code file being uploaded to the MCU from the terminal in real time, plus demonstrates some of the Words. See https://www.youtube.com/watch?v=60sNQISzEks

27 Oct 2019: A 27 second YouTube video showing the on-MCU cloning process I use. This was used to produce the 128KB STM32F103 developers image with RTS/CTS hardware handshaking below. See: https://www.youtube.com/watch?v=uWykNpPGsAg&feature=youtu.be

26 Oct 2019: A realtime video showing how fast the 128KB Developers Image (running at 72 MHz) lists all its Words.

26 Oct 2019: STM32F103 72 MHz clock Word. It utilizes a 8Mhz Crystal and the PLL. This makes the MCU NINE TIMES FASTER than the standard Mecrisp-Stellaris image.

23 Oct 2019: A 128KB STM32F103 developers image with RTS/CTS hardware handshaking. This is the same Kernel as below (20 Oct) but fully loaded with development support files. For instance every Peripheral register can be read in real time.

20 Oct 2019: Special STM32F103 image with RTS/CTS hardware handshaking. When using RTS/CTS handshaking, no terminal “End of Line Delay” is required, plus it is much faster uploading source code. This Image will work on any STM32F103 based boards with 64KB Flash. such as BluePills, Olimex-P103 etc.

10 Oct 2019: My Forth IDE, a summary.

8 Oct 2019: Cortex-M Assembly, the THUMB Instruction Set.

7 Oct 2019: Using Interactive Assembly to create a millisecond delay for Cortex-M0 and M3.

1 Oct 2019: Using Arm-none-eabi-as to create Inline Machine Code for a accurate Cortex-M0 Delay Word. This should work on other Cortex-M chips.

24 Sep 2019: Fixed Point (s31.32) Example program, STM32F0xx Baud Rate Calculator

18 Sep 2019: Fixed a bad link in: What the heck are all those LSHIFT Words?

17 Sep 2019: Updated my GREEN PILL STM32F051K8 hand soldered (under $1.50) board.

13 Sep 2019: Added “POSTPONE” example to the glossary.

12 Sep 2019: Updated my Development Environment for Forth notes.

10 Sep 2019: Programming Questions and Answers Email me your Forth questions and I’ll try to answer them here. You can also leave them on Slashdot at https://sourceforge.net/p/mecrisp-stellaris-folkdoc/discussion/ under the “Questions about programming” topic.

04 Sep 2019: Two interesting articles regarding Critical System Safety Design

01 Sep 2019: Contributing Forth Code; How to submit your code to this site for others to see or use.

31 Aug 2019: How to make your program run after a reboot or power up (TURNKEY)

30 Aug 2019: User contributed BLUE PILL board EXAMPLE CODE.

22 Aug 2019: New Page This Months HOT MCU

20 Aug 2019: Notes on atomic operations

15 Aug 2019: Blue Pill Starter Kit. Contains files to help a new Forth Blue Pill user manipulate the GPIO pins to light the LED on GPIOC-13.

12 Aug 2019: Update: hand made prototype 2.3mm copper rivet soldered in pcb

11 Aug 2019: A Interesting Forth Article “There’s Forth In That!” https://wiki.forth-ev.de/doku.php/events:ef2018:forth-in-that

11 Aug 2018: Coloured columns added to Dictionary Lister

11 Aug 2019: Colour your output

10 Aug 2019: The First Rule Of Forth.

6 Aug 2019: Added a Common User Issues Page for beginners.

4 Aug 2019: Handmade copper rivets for prototype pcb construction.

30 July 2019: Levels of Code in Forth Programming, a essay by the late Jeff Fox.

28 July 2019: A new page about CHIPS

28 July 2019: Updated DEADBUG pics added to the Prototyping Page

27 July 2019: A four column Dictionary Lister

8 July 2019: A rant about the “Blue Pill”, a ‘bargain’ STM32F103 based hobby pcb.

21 June 2019: For those wanting to get into the wonderful world of Microprocessors, I’ve created a “Learning Microprocessors” web page at https://learning-microprocessors.sourceforge.io/index.html

20 June 2019: A new release of my SVD2Forth package, svd2forth-v3-stm32.zip. This release features a 32 bit register pretty print and a STM32xxxx.svd.equ.s equates file for use with Gnu Assembler as used in the Mecrisp-Stellaris source.

8 June 2019: A new rant The Water Barons

30 May 2019: Started a C and Forth example/comparison page for common proceedures. Not much there now but it will grow in time.

26 May 2019: A blog page on Forth Sourcecode Documentation

19 May 2019: RANTS: Reinventing The Wheel, The Library Myth, Use MS Windows or Die

12 May 2019: Run Mecrisp-Stellaris-RA for a STM32F051 MCU on Linux easily without a VM or any hardware.

5 May 2019: Some general prototyping notes

30 April 2019: Matthias Koch architect of Mecrisp awarded the “Swap Dragon Prize” by the German Forth Association.

26 April 2019: April 2003 version of NASA staff’s list of the uses of Forth in space applications.

17 April 2019: Forth-sixel | terminal raster graphics and gui’s in forth by Andreas Wagner https://hub.darcs.net/pointfree/forth-sixel/ [andreas.wagner]

30 Mar 2019: Unhandled Interrupt 00000003 and how to erase with DF-UTIL

27 Mar 2019: A interesting article on Little-endian vs. big-endian

15 Mar 2019: Why are Atmel Sam series Arm chips (used by Arduino) problematic to port to Mecrisp-Stellaris ?

26 Feb 2019: A brand new 28 byte Blinky in pure Assembly Language for the STM32F0 Discovery Board by the Wizard of Assembly, Matthias Koch.

4 Feb 2019: A special svd2forth for the Blue Pill (svd2forth-v3-STM32F103C8T6-1.0.zip) that uses Jeelabs Word syntax i.e. “RCC-CR” instead of my default “RCC_CR”

16 Jan 2019: Started a new page for Fixed Point Examples(s31.32)

10 Jan 2019: Autobaud Mod for a STM32F0xx Kernel

10 Jan 2019: Colourize your terminal ERROR MESSAGES with RED for CRITICAL and BLUE for warnings. This makes Source Code errors much more visible, hence easier to see as they upload to the target.

05 Jan 2019: A New, Easy Integrated Cloning Method

02 Jan 2019: Expansion of the INLINE Word Examples in the Glossary showing INLING differences between RA and Classic Kernels.

24 Dec 2018: Matthias Koch Christmas Special: CortexM WFI Insight Words for us!

23 Dec 2018: How Does Mecrisp-Stellaris deal with Random Access Memory ? Create a Free Ram program.

20 Dec 2018: Multitasking, STMF0 Discovery Board Push Button Press Counter using Interrupts DEMO:. I hope the new layout makes it easier to grok STM32 Interrupt usage.

20 Dec 2018: Stack sizes and how to change them

20 Dec 2018: Manipulate Your Stack with Forth Wizard by Peter Sovietov

20 Dec 2018: Added RA Kernel Page. Don't have The Register Allocator Kernel for your MCU ? convert your Classic Kernel to RA by yourself!

20 DEC 2018: Roll and -Roll Words included in the RA Kernels (only).

19 Dec 2012: STMicroelectronics STM32F01x Interrupt and Exception Vectors

12 Dec 2018: Blocks, the Forth File System

10 Dec 2018: Do you have a MCU not listed in the suported hardware list but want to try Mecrisp-Stellaris on it ? See the Porters Guide.

8 Dec 2018: YouTube Videos you may find interesting.

6 Dec 2018: Word Flag Codes added to the Dictionary.

5 Dec 2018: Stepper Motor Demo, including a short YouTube Video.

4 Dec 2018: Forth for the Numworks Open Source Calculator.

2 Dec 2018: Stair Stepping, what is it, and how is it fixed ?

30 Nov 2018: A Modern Forth IDE, including a short YouTube Video.

27 Nov 2018: Added the Cornerstone Word to the Glossary with an example.

21 Nov 18: Added a svd2forth for the Nordic MCUs get it here. Release contains memmap.fs and bitfields.fs for the NRF51 and the newly supported nrf52840 MCU (24Nov2018).

20 Nov 18: All the Svd2forth tarballs are now in the one location here. They include STM32Fxx versions 1 -3, Tiva TM4C1294NCPDT and MSP430xx

18 Nov 2018: Do you have a Favorite MCU that you’d like to run Mecrisp-Stellaris on but it’s not in the supported hardware list?

18 Nov 2018: My favorite Mecrisp-Stellaris Kernel hacks

15 Nov 2018: Low Power on STM32F0

5 Nov 2018: Run Forth on a MSP430 MCU with 500 Bytes of Flash (or less) ! Mecrisp-Across Unofficial Documentation (A Tethered MSP430 Forth) added to this server [http://hightechdoc.net].

Note

The STM32F103C8T6 regpack is built from the ARM STM32F103XX CMSIS-SVD which has every peripheral possible for the STM32F103XX family. Some of these peripherals are NOT present in the 48 pin STM32F103C8T6. I have removed many of them, such as the DAC,FSMC, extra GPIO’s and Timers but if you’re familiar with the STM32F103C8T6 and find errors, please email me.

17 October 2018: Official Freenode Mecrisp-Stellaris IRC channel created: chat.freenode.net #mecrisp-stellaris

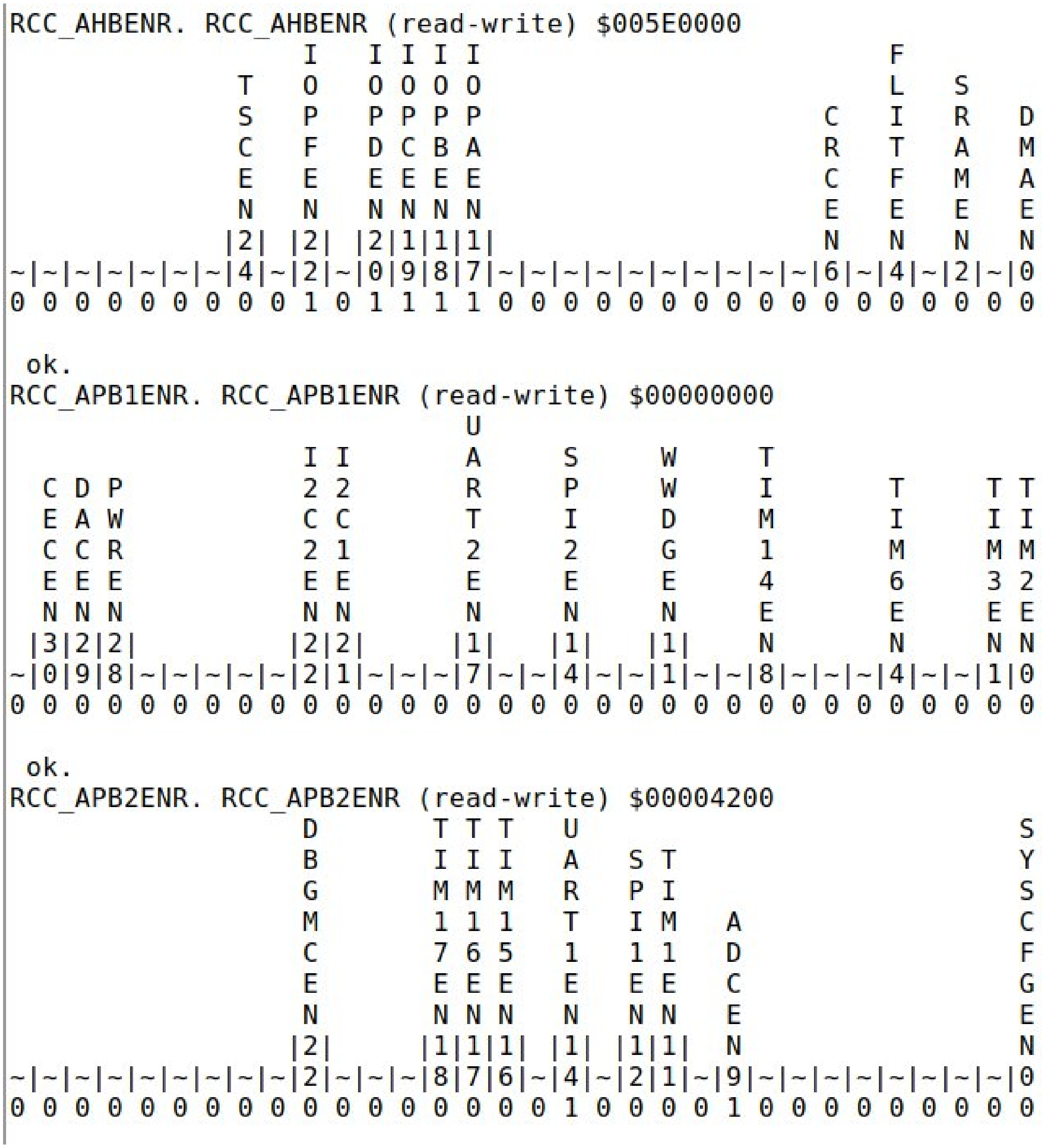
28 Jun 2018: Updated the Supported Hardware Page adding Cypress and Analog Devices MCU’s. Updated the Forth Interactive Alternatives page.

8 June 2018: Updated the Executing/Inlining Assembly/Machine Code Page with a blinky comparison, one in assembler, one in Forth, which is smallest, which is easiest to write ?

1 June 2018: Updated the Inline Assembly/Machine Code Page with a workflow example.

30 May 2018 Programming Language Benchmarks Some Speed Comparisons.

25 May 2018: Svd2forth-v3 released. Peripheral Register Pretty Printing with Legends for easier real time development and debugging.



```
RCC_AHBENR. RCC_AHBENR (read-write) $005E0000
                  I   I I I I                         F
              T   O   O O O O                         L   S
              S   P   P P P P                     C   I   R   D
              C   F   D C B A                     R   T   A   M
              E   E   E E E E                     C   F   M   A
              N   N   N N N N                     E   E   E   E
             |2| |2| |2|1|1|1|                    N   N   N   N
~|~|~|~|~|~|~|4|~|2|~|0|9|8|7|~|~|~|~|~|~|~|~|~|~|6|~|4|~|2|~|0
0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
RCC_APB1ENR. RCC_APB1ENR (read-write) $00000000
                            U
                  I I       A     S     W     T
  C D P           2 2       R     P     W     I       T     T T
  E A W           C C       T     I     D     M       I     I I
  C C R           2 1       2     2     G     1       M     M M
  E E E           E E       E     E     E     4       6     3 2
  N N N           N N       N     N     N     E       E     E E
 |3|2|2|         |2|2|     |1|   |1|   |1|    N       N     N N
~|0|9|8|~|~|~|~|~|2|1|~|~|~|7|~|~|4|~|~|1|~|~|8|~|~|~|4|~|~|1|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 ok.
RCC_APB2ENR. RCC_APB2ENR (read-write) $00004200
                  D       T T T   U                           S
                  B       I I I   A   S T                     Y
                  G       M M M   R   P I                     S
                  M       1 1 1   T   I M   A                 C
                  C       7 6 5   1   1 1   D                 F
                  E       E E E   E   E E   C                 G
                  N       N N N   N   N N   E                 E
                 |2|     |1|1|1| |1| |1|1|  N                 N
~|~|~|~|~|~|~|~|~|2|~|~|~|8|7|6|~|4|~|2|1|~|9|~|~|~|~|~|~|~|~|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
```

29 Apr 2018: A rewritten, stand alone 460800 Baud 48Mhz clock for the STM32F0 Discovery Board. It uses the Xtal Clock from the SWD subsystem but can easily be configured to use the onboard RC 8 MHz Internal Clock for any STM32F0xx.

27 Apr 2018: A ready built Kernel with CTS hardware handshaking at 406800 Baud for hardware Sourcecode Error Detection.

14 Apr 2018: Added a Compiler Message Table to the Dictionary page.

24 Mar 2018: Get a Nokia 5110 lcd display up and running, to ‘all pixels on’ test stage using Mecrisp-Stellaris, with code and Logic Analyser pictures of the data and control signals.

19 Mar 2018: Potentiometer driven comparator Window Mode Example on a STM32F0 Disco Board. The code runs standalone.

17 Mar 2018: Update to the 1mS Timer, it’s now standalone. Used to time software and hardware events.

16 Mar 2018: Code Reviews and Libraries, a short Rant.

25 Feb 2018: A Sunrise and Sunset Calculator project by Andrew Palm.

25 Feb 2018: A square root, trig, and inverse trig functions for 64bit Fixed point (s31.32) numbers library by Andrew Palm.

21 Dec 2017: A STM32F0 Discovery Board Interrupt Program using the USER pushbutton. It’s standalone, upload and run it.

19 Oct 2017: What are the Forth (interactive) Embedded Alternatives on Cortex M and how much Flash do they require to run ?

29 Sep 2017: Updates to the Example STM32 Boards supported by Mecrisp-Stellaris page. New boards and more

pics added.

28 Sep 2017: Updated the how to flash Mecrisp-Stellaris page.

26 Sep 2017: Do you use MSP430 ? Svd2mecrisp is now available for MSP430xx.

17 Sep 2017: Svd2forth-v1 uses a ARM CMSIS-SVD XML file, to produce a Mecrisp-Stellaris Memory Map and Register reference file for a STM32Fx MCU. Svd2forth-v1 creates bare bones, minimal files.

12 Sept 2017: Manfred Mahlow’s VOCS is included in the latest Mecrisp-Stellaris release.

09 Sept 2017: Analog To digital 19ch scanner. Version 2, is a ADC all channel scanner, no DMA, no interrupts, using only a spin delay to wait for the end of conversion sequence(s). It works by scanning each channel and pushing the results on the Stack. After a conversion sequence of 19 channels is finished, it pops the results off the stack, scales and displays them.

01 Sept 2017: Interactive Assembly Language

21 August 2017: How to clone a Mecrisp-Stellaris image from a existing STM32F MCU or Board

27 July 2017:Vocabularies , details and demo by Manfred Mahlow

22 July 2017: A **`EPUB ebook`** of this site, (generated automatically by Sphinx). File URL’s won’t resolve as it’s a ebook, but everything else should be ok.

22 July 2017: This example prints some GPIO Modes, i.e. IN, OUT, ANALOG etc, to demonstrate the versatility of <builds does>, often described as the ‘jewel of forth’.

15 July 2017: Benchmark your Forth MCU. Confirm that a standard STM32F051 is **6.7** times faster** than a Mega328 as used by Arduino, or **13.3** times faster when overclocked!

12 July 2017: Svd2forth-v2 updated (release-6ba968d6f6). In addition to *Peripheral print Words*, there are now individual *Register print Words* and *Register BitField fetch Words*.

04 July 2017: Push button switch debounce demo

02 July 2017: A new Svd2gas assembler version and the *smallest* Blinky binary file we can make, just *32 bytes*.

16 June 2017: Mecrisp-Stellaris Interrupt mechanism documentation *updated*.

June-2017: A video page, for interesting and informative videos about Mecrisp-Stellaris.

May-2017: Mecrisp-Stellaris comes with a set of Common Programs, this work in progress attempts to describe them.

May-2017: Forth Development Tips

May-2017: Visualising The Forth Stack, or “The Stack is doing my head in !”

May-2017: A Forth loadmeter

May-2017: Kernel patches; modify the Mecrisp-Stellaris source, change the terminal USART, add hardware handshaking, simplify the Stack print word “.s”

May-2017: A (small) library of development words

May-2017: Writing STM32 assembly? … Svd2gas is designed to handle register names and memory mapping, to make your coding easier and less error prone

May-2017, Updated: hardware flow control for your serial terminal

April-2017: Seven blinkies; simple spinners, multi-tasking, interrupt driven and one assembly language Blinky. If you’d like to see another kind, just contact me

KEIL C V/S SVD2FORTH

Peripheral Programming.

This page attempts to show how both the latest Keil C Programming Language and Forth approach the process of programming Cortex-M0 peripherals. I will try and illustrate the differences and similarities of both methods using a real world example.

Being familiar with the Cortex-M0 hardware will make the article easier to follow.

C Programming Language

Keil CMSIS include and header files are used for the examples. See: Keil.STM32F0xx_DFP.2.0.0.pack

```
*************************************************************
*****************
* @file    stm32f051x8.h
* @author  MCD Application Team
* @brief   CMSIS Cortex-M0 Device Peripheral Access Layer
Header File.
*          This file contains all the peripheral register's
definitions, bits
*          definitions and memory mapping for STM32F0xx
devices.
*
*          This file contains:
*           - Data structures and the address mapping for all
peripherals
*           - Peripheral's registers declarations and bits
definition
*           - Macros to access peripherals registers hardware
*
*************************************************************
*****************
```

```
* @attention
*
* <h2><center>&copy; COPYRIGHT(c) 2016
STMicroelectronics</center></h2>
*
* Redistribution and use in source and binary forms, with or
without modification,
* are permitted provided that the following conditions are
met:
*   1. Redistributions of source code must retain the above
copyright notice,
*      this list of conditions and the following disclaimer.
*   2. Redistributions in binary form must reproduce the
above copyright notice,
*      this list of conditions and the following disclaimer
in the documentation
*      and/or other materials provided with the distribution.
*   3. Neither the name of STMicroelectronics nor the names
of its contributors
*      may be used to endorse or promote products derived
from this software
*      without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR
* SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
*
**************************************************************
```

```
*****************
*/
```

Forth Programming Language

Mecrisp-Stellaris with Svd2forth is used, both are licensed under the GPLV3.

Hardware Documentation

Not all peripherals used are shown but the full reference manual can be downloaded here **`Download: STM32F0xx Reference Manual (RM0091)`**

RCC Peripheral

Embedded design mainly involves configuring, writing and reading data to and from peripheral devices. The example used here is the RCC peripheral which controls the MCU Clock and enables/disables peripheral devices etc. This is a key Cortex-M peripheral.

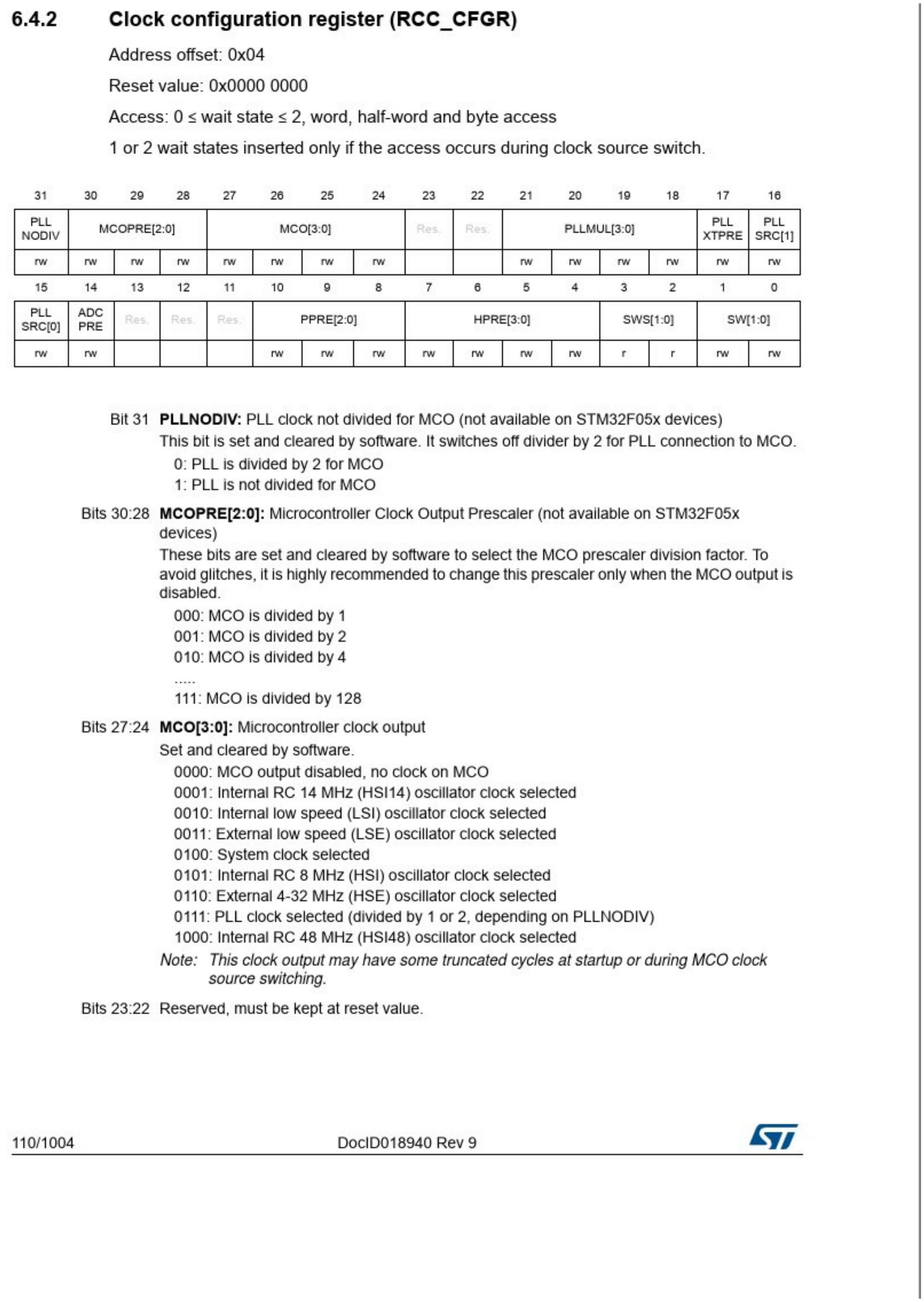
RCC_CFGR Register

We will concentrate on the RCC_CFGR register which is used to control the MCU main clock.

These three STMicro Reference Manual (RM0091) pages describe RCC_CFGR register operation in detail.

RCC_CFGR register page 1

Reset and clock control (RCC) RM0091



6.4.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLL NODIV	MCOPRE[2:0]			MCO[3:0]				Res.	Res.	PLLMUL[3:0]				PLL XTPRE	PLL SRC[1]
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLL SRC[0]	ADC PRE	Res.	Res.	Res.	PPRE[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw				rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bit 31 **PLLNODIV:** PLL clock not divided for MCO (not available on STM32F05x devices)

This bit is set and cleared by software. It switches off divider by 2 for PLL connection to MCO.

0: PLL is divided by 2 for MCO
1: PLL is not divided for MCO

Bits 30:28 **MCOPRE[2:0]:** Microcontroller Clock Output Prescaler (not available on STM32F05x devices)

These bits are set and cleared by software to select the MCO prescaler division factor. To avoid glitches, it is highly recommended to change this prescaler only when the MCO output is disabled.

000: MCO is divided by 1
001: MCO is divided by 2
010: MCO is divided by 4
.....
111: MCO is divided by 128

Bits 27:24 **MCO[3:0]:** Microcontroller clock output

Set and cleared by software.

0000: MCO output disabled, no clock on MCO
0001: Internal RC 14 MHz (HSI14) oscillator clock selected
0010: Internal low speed (LSI) oscillator clock selected
0011: External low speed (LSE) oscillator clock selected
0100: System clock selected
0101: Internal RC 8 MHz (HSI) oscillator clock selected
0110: External 4-32 MHz (HSE) oscillator clock selected
0111: PLL clock selected (divided by 1 or 2, depending on PLLNODIV)
1000: Internal RC 48 MHz (HSI48) oscillator clock selected

Note: This clock output may have some truncated cycles at startup or during MCO clock source switching.

Bits 23:22 Reserved, must be kept at reset value.

RCC_CFGR register page 2

Bits 21:18 **PLLMUL[3:0]:** PLL multiplication factor

These bits are written by software to define the PLL multiplication factor. These bits can be written only when PLL is disabled.

Caution: The PLL output frequency must not exceed 48 MHz.

0000: PLL input clock x 2
0001: PLL input clock x 3
0010: PLL input clock x 4
0011: PLL input clock x 5
0100: PLL input clock x 6
0101: PLL input clock x 7
0110: PLL input clock x 8
0111: PLL input clock x 9
1000: PLL input clock x 10
1001: PLL input clock x 11
1010: PLL input clock x 12
1011: PLL input clock x 13
1100: PLL input clock x 14
1101: PLL input clock x 15
1110: PLL input clock x 16
1111: PLL input clock x 16

Bit 17 **PLLXTPRE**: HSE divider for PLL input clock

This bit is the same bit as bit PREDIV[0] from RCC_CFGR2. Refer to RCC_CFGR2 PREDIV bits description for its meaning.

Bits 16:15 **PLLSRC[1:0]:** PLL input clock source

Set and cleared by software to select PLL or PREDIV clock source. These bits can be written only when PLL is disabled.

00: HSI/2 selected as PLL input clock (PREDIV forced to divide by 2 on STM32F04x, STM32F07x and STM32F09x devices)
01: HSI/PREDIV selected as PLL input clock
10: HSE/PREDIV selected as PLL input clock
11: HSI48/PREDIV selected as PLL input clock

Note: Bit PLLSRC[0] is available only on STM32F04x, STM32F07x and STM32F09x devices, otherwise it is reserved (with value zero).

Bit 14 **ADCPRE:** ADC prescaler

Obsolete setting. Proper ADC clock selection is done inside the ADC_CFGR2 (refer to *Section 13.12.5: ADC configuration register 2 (ADC_CFGR2) on page 263*).

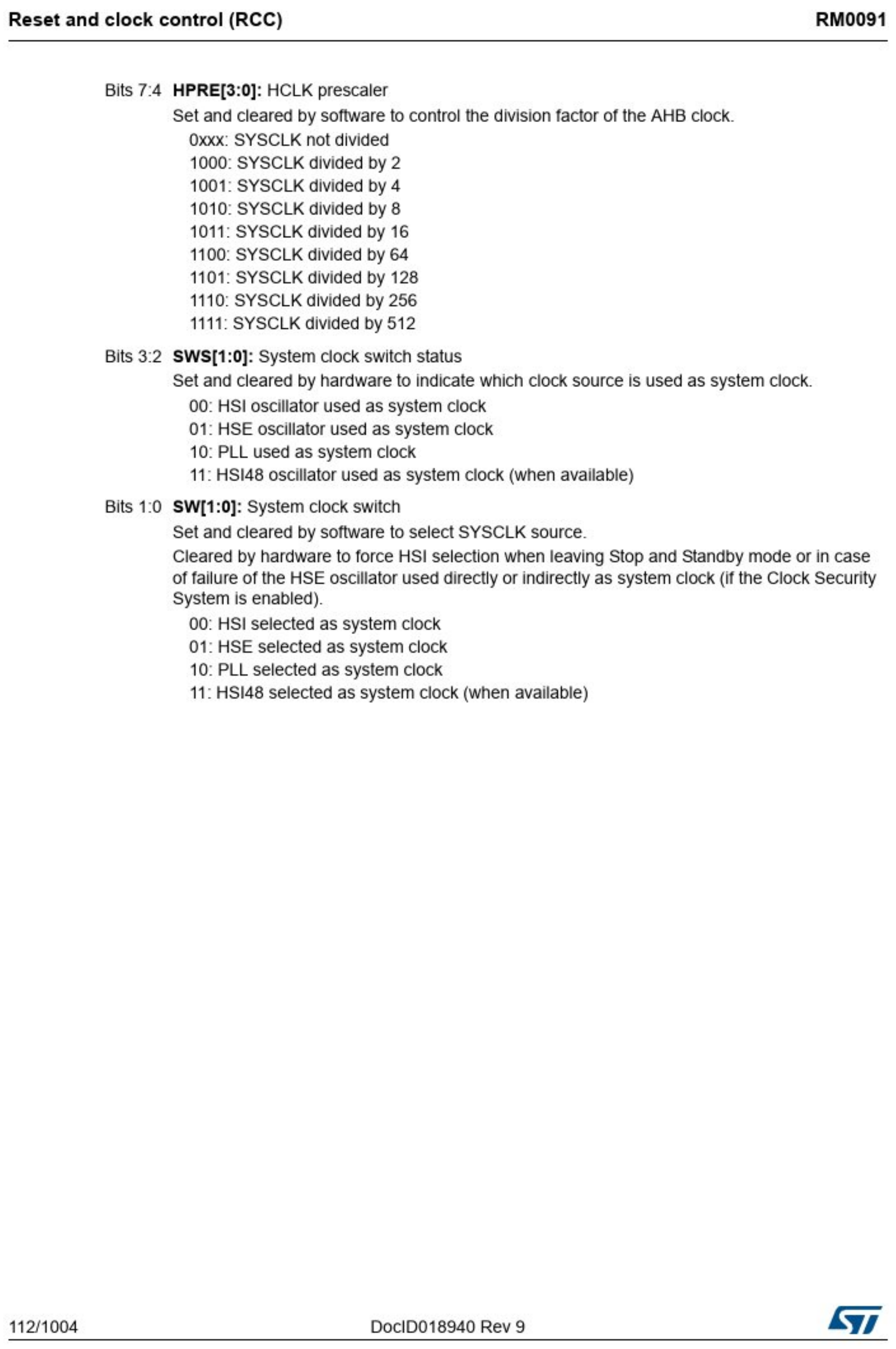
Bits 13:11 Reserved, must be kept at reset value.

Bits 10:8 **PPRE[2:0]:** PCLK prescaler

Set and cleared by software to control the division factor of the APB clock (PCLK).

0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

RCC_CFGR register page 3



Bits 7:4 **HPRE[3:0]:** HCLK prescaler

Set and cleared by software to control the division factor of the AHB clock.

0xxx: SYSCLK not divided
1000: SYSCLK divided by 2
1001: SYSCLK divided by 4
1010: SYSCLK divided by 8
1011: SYSCLK divided by 16
1100: SYSCLK divided by 64
1101: SYSCLK divided by 128
1110: SYSCLK divided by 256
1111: SYSCLK divided by 512

Bits 3:2 **SWS[1:0]:** System clock switch status

Set and cleared by hardware to indicate which clock source is used as system clock.

00: HSI oscillator used as system clock
01: HSE oscillator used as system clock
10: PLL used as system clock
11: HSI48 oscillator used as system clock (when available)

Bits 1:0 **SW[1:0]:** System clock switch

Set and cleared by software to select SYSCLK source.

Cleared by hardware to force HSI selection when leaving Stop and Standby mode or in case of failure of the HSE oscillator used directly or indirectly as system clock (if the Clock Security System is enabled).

00: HSI selected as system clock
01: HSE selected as system clock
10: PLL selected as system clock
11: HSI48 selected as system clock (when available)

RCC_CFGR Configuration Example

Printout of the live RCC_CFGR configuration taken from my development hardware.

```
RCC_CFGR.   $001D040A
                            P P P
                            L L L
                            L L L A
                            X S S D
                            T R R C
                            P C C P
       |   MCO  |    |PLLMUL |R 1 0 R|      |PPRE |
       |3:2:1:0|    |3:2:1:0|E | |  E|      |2:1:0|   HPRE
| SWS| SW
       |2|2|2|2|    |2|2|1|1|1|1|1|1|      |1|  |
|3:2:1:0|1:0|1:0
~|~|~|~|7|6|5|4|~|~|1|0|9|8|7|6|5|4|~|~|~|0|9|8|7|6|5|4|3|2|1
|0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1
0
```

Configuration Details

Note: Only relevant bitfields are listed below.

Bitfield	Binary Value	Description
SW	10	PLL selected as system clock
SWS	10	PLL used as system clock

Bitfield	**Binary Value**	**Description**
HPRE	0xxx	SYSCLK not divided
PPRE	100	HCLK divided by 2
PLLSRC	10	HSE/PREDIV selected as PLL input clock
PLLMUL	0111	PLL input clock x 9
MCO	000	MCO is divided by 1

Configuration Summary

The clock (HPRE) uses the PLL with a undivided 9x multiplier. As the HPRE clock is 8 MHz multiplying by nine = 72 MHz. This MCU is being *overclocked* as the spec gives a maximum clock of 48 MHz for the Cortex-M0 family.

Note

I have run one unit at 96MHz for 12 months with no ill effects. I only use this for development to miniminse delays such as serial uploading. As SYSTICK is used for all timing, the source requires no changes when altering the clock speed.

Peripheral memory map

Forth

Forth Memaps and Bitfields are derived from the CMSIS-SVD and transformed by XSLT with Svd2forth.

```
$40021000 constant RCC              \ Reset and clock control
RCC $4 + constant RCC_CFGR (  )     \ Clock configuration
register  RCC_CFGR
```

C Programming Language

```
#define PERIPH_BASE           ((uint32_t)0x40000000U)
/*!< Peripheral base address in the alias region */
#define AHBPERIPH_BASE        (PERIPH_BASE + 0x00020000)
#define RCC_BASE              (AHBPERIPH_BASE + 0x00001000)
```

Summary

Both systems are much the same at the peripheral memory map stage.

RCC_CFGR Bitfield Definitions

Note the Stack comments, these are my latest version and should be correct. “b” is a binary number 0|1.

Forth

```
\ RCC_CFGR ()
: RCC_CFGR_SW ( %bb -- x addr ) RCC_CFGR ;                  \
RCC_CFGR_SW, System clock Switch
: RCC_CFGR_SWS ( %bb -- x addr ) 2 lshift RCC_CFGR ;        \
RCC_CFGR_SWS, System Clock Switch Status
: RCC_CFGR_HPRE ( %bbbb -- x addr ) 4 lshift RCC_CFGR ;     \
RCC_CFGR_HPRE, AHB prescaler
: RCC_CFGR_PPRE ( %bbb -- x addr ) 8 lshift RCC_CFGR ;      \
RCC_CFGR_PPRE, APB Low speed prescaler  APB1
: RCC_CFGR_ADCPRE ( %bb -- x addr ) 14 lshift RCC_CFGR ;   \
RCC_CFGR_ADCPRE, ADC prescaler
: RCC_CFGR_PLLSRC ( %bb -- x addr ) 15 lshift RCC_CFGR ;   \
RCC_CFGR_PLLSRC, PLL entry clock source
: RCC_CFGR_PLLXTPRE ( -- x addr ) 17 bit RCC_CFGR ;        \
RCC_CFGR_PLLXTPRE, HSE divider for PLL entry
: RCC_CFGR_PLLMUL ( %bbbb -- x addr ) 18 lshift RCC_CFGR ; \
RCC_CFGR_PLLMUL, PLL Multiplication Factor
: RCC_CFGR_MCO ( %bbb -- x addr ) 24 lshift RCC_CFGR ;     \
RCC_CFGR_MCO, Microcontroller clock  output
```

Forth Configuration

To achieve the example configuration.

```
\ f0-disco-72mhz: Set clock to 72MHz

$40021000 constant RCC                  \ Reset and clock
control
RCC $0 + constant RCC_CR (  )           \ Clock control
register
RCC $4 + constant RCC_CFGR (  )         \ Clock configuration
register  RCC_CFGR

$40022000 constant Flash                \ When raising the
Clock speed, the Flash LATENCY must be extended.
Flash $0 + constant Flash_ACR (  )      \ Flash access control
register

: bit ( u -- u ) 1 swap lshift  1-foldable ;    \ turn a bit
position into a binary number.

: Flash_ACR_LATENCY ( %bbb -- x addr ) Flash_ACR ;         \
Flash_ACR_LATENCY, LATENCY
: Flash_ACR_PRFTBE ( -- x addr ) 4 bit Flash_ACR ;         \
PRFTBE Prefetch buffer enable
: RCC_CR_HSEON ( -- x addr ) 16 bit RCC_CR ;               \
External High Speed clock  enable
: RCC_CR_HSERDY? ( -- 1|0 ) 17 bit RCC_CR bit@ ;           \
RCC_CR_HSERDY, External High Speed clock ready  flag
: RCC_CFGR_PLLMUL ( %bbbb -- x addr ) 18 lshift RCC_CFGR ; \
PLL Multiplication Factor
```

```
: RCC_CFGR_PLLSRC ( %bb -- x addr ) 15 lshift RCC_CFGR ;   \
PLL entry clock source
: RCC_CR_PLLON ( -- x addr ) 24 bit RCC_CR ;               \
PLL enable
: RCC_CR_PLLRDY? ( -- 1|0 ) 25 bit RCC_CR bit@ ;           \
RCC_CR_PLLRDY, PLL clock ready flag
: RCC_CFGR_SW ( %bb -- x addr ) RCC_CFGR ;                 \
System clock Switch
: RCC_CFGR_PPRE ( %bbb -- x addr ) 8 lshift RCC_CFGR ;     \
APB Low speed prescaler  APB1
: RCC_CFGR_HPRE ( %bbbb -- x addr ) 4 lshift RCC_CFGR ;    \
AHB prescaler

: f0-disco-72mhz
\ Set flash wait states to 1, turn on prefetch bit. One wait
state, if 24 MHz < SYSCLK  48 MHz
%011  Flash_ACR_LATENCY bis!    \ LATENCY --> max = 1 for the
stm32f0, note more latency uses less power
Flash_ACR_PRFTBE bis!

\ Enable HSE oscillator
RCC_CR_HSEON bis!
BEGIN  RCC_CR_HSERDY? UNTIL     \ Is HSE ready ?

\ Set PLL multiplication factor to 8, clock is external HSE
8MHz Xtal on Disco board.
%111 RCC_CFGR_PLLMUL bis!       \ %111: PLL input clock x 9
%10 RCC_CFGR_PLLSRC bis!        \ %10: HSE/PREDIV, 01:
HSI/PREDIV

\ Enable PLL
RCC_CR_PLLON bis!
BEGIN RCC_CR_PLLRDY? UNTIL      \ Is PLL ready ?

\ Switch to PLL, set ABP and AHB Prescalers
%10 RCC_CFGR_SW bis!            \ System clock Switch: %10
PLL selected as system clock
%100  RCC_CFGR_PPRE bis!        \ APB Low speed prescaler
APB1 (pclock): %100: HCLK divided by 2
;

f0-disco-72mhz
```

C Programming Language

Peripheral register structure

```
/**
  * @brief Reset and Clock Control
  */

typedef struct
{
  __IO uint32_t CR;         /*!< RCC clock control register,
Address offset: 0x00 */
  __IO uint32_t CFGR;       /*!< RCC clock configuration
register,              Address offset: 0x04 */
  __IO uint32_t CIR;        /*!< RCC clock interrupt
register,                  Address offset: 0x08 */
  __IO uint32_t APB2RSTR;   /*!< RCC APB2 peripheral reset
register,            Address offset: 0x0C */
  __IO uint32_t APB1RSTR;   /*!< RCC APB1 peripheral reset
register,            Address offset: 0x10 */
  __IO uint32_t AHBENR;     /*!< RCC AHB peripheral clock
register,             Address offset: 0x14 */
  __IO uint32_t APB2ENR;    /*!< RCC APB2 peripheral clock
enable register,     Address offset: 0x18 */
  __IO uint32_t APB1ENR;    /*!< RCC APB1 peripheral clock
enable register,     Address offset: 0x1C */
  __IO uint32_t BDCR;       /*!< RCC Backup domain control
register,            Address offset: 0x20 */
  __IO uint32_t CSR;        /*!< RCC clock control & status
register,           Address offset: 0x24 */
  __IO uint32_t AHBRSTR;    /*!< RCC AHB peripheral reset
register,             Address offset: 0x28 */
  __IO uint32_t CFGR2;      /*!< RCC clock configuration
register 2,            Address offset: 0x2C */
  __IO uint32_t CFGR3;      /*!< RCC clock configuration
register 3,            Address offset: 0x30 */
  __IO uint32_t CR2;        /*!< RCC clock control register
2,                  Address offset: 0x34 */
} RCC_TypeDef;
```

Peripheral_declaration

```
#define RCC                ((RCC_TypeDef *) RCC_BASE)
```

Bit Definitions

```
/*!< SW configuration */
#define RCC_CFGR_SW_Pos                          (0U)
#define RCC_CFGR_SW_Msk                          (0x3U <<
RCC_CFGR_SW_Pos)     /*!< 0x00000003 */
#define RCC_CFGR_SW
RCC_CFGR_SW_Msk              /*!< SW[1:0] bits (System clock
Switch) */
#define RCC_CFGR_SW_0                            (0x1U <<
RCC_CFGR_SW_Pos)     /*!< 0x00000001 */
#define RCC_CFGR_SW_1                            (0x2U <<
RCC_CFGR_SW_Pos)     /*!< 0x00000002 */

#define RCC_CFGR_SW_HSI
(0x00000000U)                /*!< HSI selected as system
clock */
#define RCC_CFGR_SW_HSE
(0x00000001U)                /*!< HSE selected as system
clock */
#define RCC_CFGR_SW_PLL
(0x00000002U)                /*!< PLL selected as system
clock */

/*!< SWS configuration */
#define RCC_CFGR_SWS_Pos                         (2U)
#define RCC_CFGR_SWS_Msk                         (0x3U <<
RCC_CFGR_SWS_Pos)    /*!< 0x0000000C */
#define RCC_CFGR_SWS
RCC_CFGR_SWS_Msk             /*!< SWS[1:0] bits (System
Clock Switch Status) */
#define RCC_CFGR_SWS_0                           (0x1U <<
RCC_CFGR_SWS_Pos)    /*!< 0x00000004 */
#define RCC_CFGR_SWS_1                           (0x2U <<
RCC_CFGR_SWS_Pos)    /*!< 0x00000008 */

#define RCC_CFGR_SWS_HSI
(0x00000000U)                /*!< HSI oscillator used as
system clock */
#define RCC_CFGR_SWS_HSE
(0x00000004U)                /*!< HSE oscillator used as
system clock */
#define RCC_CFGR_SWS_PLL
(0x00000008U)                /*!< PLL used as system clock
*/

/*!< HPRE configuration */
```

```
#define RCC_CFGR_HPRE_Pos                               (4U)
#define RCC_CFGR_HPRE_Msk                               (0xFU <<
RCC_CFGR_HPRE_Pos)   /*!< 0x000000F0 */
#define RCC_CFGR_HPRE
RCC_CFGR_HPRE_Msk             /*!< HPRE[3:0] bits (AHB
prescaler) */
#define RCC_CFGR_HPRE_0                                 (0x1U <<
RCC_CFGR_HPRE_Pos)   /*!< 0x00000010 */
#define RCC_CFGR_HPRE_1                                 (0x2U <<
RCC_CFGR_HPRE_Pos)   /*!< 0x00000020 */
#define RCC_CFGR_HPRE_2                                 (0x4U <<
RCC_CFGR_HPRE_Pos)   /*!< 0x00000040 */
#define RCC_CFGR_HPRE_3                                 (0x8U <<
RCC_CFGR_HPRE_Pos)   /*!< 0x00000080 */

#define RCC_CFGR_HPRE_DIV1
(0x00000000U)                 /*!< SYSCLK not divided */
#define RCC_CFGR_HPRE_DIV2
(0x00000080U)                 /*!< SYSCLK divided by 2 */
#define RCC_CFGR_HPRE_DIV4
(0x00000090U)                 /*!< SYSCLK divided by 4 */
#define RCC_CFGR_HPRE_DIV8
(0x000000A0U)                 /*!< SYSCLK divided by 8 */
#define RCC_CFGR_HPRE_DIV16
(0x000000B0U)                 /*!< SYSCLK divided by 16 */
#define RCC_CFGR_HPRE_DIV64
(0x000000C0U)                 /*!< SYSCLK divided by 64 */
#define RCC_CFGR_HPRE_DIV128
(0x000000D0U)                 /*!< SYSCLK divided by 128 */
#define RCC_CFGR_HPRE_DIV256
(0x000000E0U)                 /*!< SYSCLK divided by 256 */
#define RCC_CFGR_HPRE_DIV512
(0x000000F0U)                 /*!< SYSCLK divided by 512 */

/*!< PPRE configuration */
#define RCC_CFGR_PPRE_Pos                               (8U)
#define RCC_CFGR_PPRE_Msk                               (0x7U <<
RCC_CFGR_PPRE_Pos)   /*!< 0x00000700 */
#define RCC_CFGR_PPRE
RCC_CFGR_PPRE_Msk             /*!< PRE[2:0] bits (APB
prescaler) */
#define RCC_CFGR_PPRE_0                                 (0x1U <<
RCC_CFGR_PPRE_Pos)   /*!< 0x00000100 */
#define RCC_CFGR_PPRE_1                                 (0x2U <<
RCC_CFGR_PPRE_Pos)   /*!< 0x00000200 */
```

```
#define RCC_CFGR_PPRE_2                          (0x4U <<
RCC_CFGR_PPRE_Pos)   /*!< 0x00000400 */

#define RCC_CFGR_PPRE_DIV1
(0x00000000U)                /*!< HCLK not divided */
#define RCC_CFGR_PPRE_DIV2_Pos                   (10U)
#define RCC_CFGR_PPRE_DIV2_Msk                   (0x1U <<
RCC_CFGR_PPRE_DIV2_Pos) /*!< 0x00000400 */
#define RCC_CFGR_PPRE_DIV2
RCC_CFGR_PPRE_DIV2_Msk       /*!< HCLK divided by 2 */
#define RCC_CFGR_PPRE_DIV4_Pos                   (8U)
#define RCC_CFGR_PPRE_DIV4_Msk                   (0x5U <<
RCC_CFGR_PPRE_DIV4_Pos) /*!< 0x00000500 */
#define RCC_CFGR_PPRE_DIV4
RCC_CFGR_PPRE_DIV4_Msk       /*!< HCLK divided by 4 */
#define RCC_CFGR_PPRE_DIV8_Pos                   (9U)
#define RCC_CFGR_PPRE_DIV8_Msk                   (0x3U <<
RCC_CFGR_PPRE_DIV8_Pos) /*!< 0x00000600 */
#define RCC_CFGR_PPRE_DIV8
RCC_CFGR_PPRE_DIV8_Msk       /*!< HCLK divided by 8 */
#define RCC_CFGR_PPRE_DIV16_Pos                  (8U)
#define RCC_CFGR_PPRE_DIV16_Msk                  (0x7U <<
RCC_CFGR_PPRE_DIV16_Pos) /*!< 0x00000700 */
#define RCC_CFGR_PPRE_DIV16
RCC_CFGR_PPRE_DIV16_Msk      /*!< HCLK divided by 16 */

/*!< ADCPPRE configuration */
#define RCC_CFGR_ADCPRE_Pos                      (14U)
#define RCC_CFGR_ADCPRE_Msk                      (0x1U <<
RCC_CFGR_ADCPRE_Pos) /*!< 0x00004000 */
#define RCC_CFGR_ADCPRE
RCC_CFGR_ADCPRE_Msk          /*!< ADCPRE bit (ADC prescaler)
*/

#define RCC_CFGR_ADCPRE_DIV2
(0x00000000U)                /*!< PCLK divided by 2 */
#define RCC_CFGR_ADCPRE_DIV4
(0x00004000U)                /*!< PCLK divided by 4 */

#define RCC_CFGR_PLLSRC_Pos                      (16U)
#define RCC_CFGR_PLLSRC_Msk                      (0x1U <<
RCC_CFGR_PLLSRC_Pos) /*!< 0x00010000 */
#define RCC_CFGR_PLLSRC
RCC_CFGR_PLLSRC_Msk          /*!< PLL entry clock source */
#define RCC_CFGR_PLLSRC_HSI_DIV2
```

```
(0x00000000U)                 /*!< HSI clock divided by 2
selected as PLL entry clock source */
#define RCC_CFGR_PLLSRC_HSE_PREDIV
(0x00010000U)                 /*!< HSE/PREDIV clock selected
as PLL entry clock source */

#define RCC_CFGR_PLLXTPRE_Pos                    (17U)
#define RCC_CFGR_PLLXTPRE_Msk                    (0x1U <<
RCC_CFGR_PLLXTPRE_Pos) /*!< 0x00020000 */
#define RCC_CFGR_PLLXTPRE
RCC_CFGR_PLLXTPRE_Msk         /*!< HSE divider for PLL entry
*/
#define RCC_CFGR_PLLXTPRE_HSE_PREDIV_DIV1
(0x00000000U)                 /*!< HSE/PREDIV clock not
divided for PLL entry */
#define RCC_CFGR_PLLXTPRE_HSE_PREDIV_DIV2
(0x00020000U)                 /*!< HSE/PREDIV clock divided
by 2 for PLL entry */

/*!< PLLMUL configuration */
#define RCC_CFGR_PLLMUL_Pos                      (18U)
#define RCC_CFGR_PLLMUL_Msk                      (0xFU <<
RCC_CFGR_PLLMUL_Pos) /*!< 0x003C0000 */
#define RCC_CFGR_PLLMUL
RCC_CFGR_PLLMUL_Msk           /*!< PLLMUL[3:0] bits (PLL
multiplication factor) */
#define RCC_CFGR_PLLMUL_0                        (0x1U <<
RCC_CFGR_PLLMUL_Pos) /*!< 0x00040000 */
#define RCC_CFGR_PLLMUL_1                        (0x2U <<
RCC_CFGR_PLLMUL_Pos) /*!< 0x00080000 */
#define RCC_CFGR_PLLMUL_2                        (0x4U <<
RCC_CFGR_PLLMUL_Pos) /*!< 0x00100000 */
#define RCC_CFGR_PLLMUL_3                        (0x8U <<
RCC_CFGR_PLLMUL_Pos) /*!< 0x00200000 */

#define RCC_CFGR_PLLMUL2
(0x00000000U)                 /*!< PLL input clock*2 */
#define RCC_CFGR_PLLMUL3
(0x00040000U)                 /*!< PLL input clock*3 */
#define RCC_CFGR_PLLMUL4
(0x00080000U)                 /*!< PLL input clock*4 */
#define RCC_CFGR_PLLMUL5
(0x000C0000U)                 /*!< PLL input clock*5 */
#define RCC_CFGR_PLLMUL6
(0x00100000U)                 /*!< PLL input clock*6 */
```

```
#define RCC_CFGR_PLLMUL7
(0x00140000U)                /*!< PLL input clock*7 */
#define RCC_CFGR_PLLMUL8
(0x00180000U)                /*!< PLL input clock*8 */
#define RCC_CFGR_PLLMUL9
(0x001C0000U)                /*!< PLL input clock*9 */
#define RCC_CFGR_PLLMUL10
(0x00200000U)                /*!< PLL input clock10 */
#define RCC_CFGR_PLLMUL11
(0x00240000U)                /*!< PLL input clock*11 */
#define RCC_CFGR_PLLMUL12
(0x00280000U)                /*!< PLL input clock*12 */
#define RCC_CFGR_PLLMUL13
(0x002C0000U)                /*!< PLL input clock*13 */
#define RCC_CFGR_PLLMUL14
(0x00300000U)                /*!< PLL input clock*14 */
#define RCC_CFGR_PLLMUL15
(0x00340000U)                /*!< PLL input clock*15 */
#define RCC_CFGR_PLLMUL16
(0x00380000U)                /*!< PLL input clock*16 */
```

Summary

keil/Drivers/CMSIS/Device/ST/STM32F0xx/Source/Templates/system_stm32f0xx.c

Quite large differences here. C uses structures and predefines every #defined legal bitfield variation ?

Keil C Configuration

Unfortunately I couldn’t find an example config the same as mine above but this one is similar (PLL and Flash are not used) and should help to illustrate the Keil C and Forth differences regarding peripheral configuration.

This is quite long as it also caters for other slightly different models in the F0 family.

```
/**

*************************************************************
*****************
  * @file    system_stm32f0xx.c
  * @author  MCD Application Team
  * @brief   CMSIS Cortex-M0 Device Peripheral Access Layer
System Source File.
  *
  * 1. This file provides two functions and one global
variable to be called from
  *    user application:
  *      - SystemInit(): This function is called at startup
just after reset and
  *                      before branch to main program. This
call is made inside
  *                      the "startup_stm32f0xx.s" file.
  *
  *      - SystemCoreClock variable: Contains the core clock
(HCLK), it can be used
  *                                  by the user application
to setup the SysTick
  *                                  timer or configure other
parameters.
  *
  *      - SystemCoreClockUpdate(): Updates the variable
SystemCoreClock and must
  *                                 be called whenever the
core clock is changed
  *                                 during program execution.
  *
  * 2. After each device reset the HSI (8 MHz) is used as
system clock source.
  *    Then SystemInit() function is called, in
"startup_stm32f0xx.s" file, to
  *    configure the system clock before to branch to main
program.
  *
  * 3. This file configures the system clock as follows:

*============================================================
=================
  *                         Supported STM32F0xx device
  *----------------------------------------------------------
-------------------
```

```
  *        System Clock source                    | HSI
  *-----------------------------------------------------------------------------
  *        SYSCLK(Hz)                             | 8000000
  *-----------------------------------------------------------------------------
  *        HCLK(Hz)                               | 8000000
  *-----------------------------------------------------------------------------
  *        AHB Prescaler                          | 1
  *-----------------------------------------------------------------------------
  *        APB1 Prescaler                         | 1
  *-----------------------------------------------------------------------------

*=============================================================================

******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2016 STMicroelectronics</center></h2>
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
  *

*************************************************************
*****************
  */

/** @addtogroup CMSIS
  * @{
  */

/** @addtogroup stm32f0xx_system
  * @{
  */

/** @addtogroup STM32F0xx_System_Private_Includes
  * @{
  */

#include "stm32f0xx.h"

/**
  * @}
  */
```

```
/** @addtogroup STM32F0xx_System_Private_TypesDefinitions
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F0xx_System_Private_Defines
  * @{
  */
#if !defined  (HSE_VALUE)
  #define HSE_VALUE    ((uint32_t)8000000) /*!< Default value
of the External oscillator in Hz.
                                                This value
can be provided and adapted by the user application. */
#endif /* HSE_VALUE */

#if !defined  (HSI_VALUE)
  #define HSI_VALUE    ((uint32_t)8000000) /*!< Default value
of the Internal oscillator in Hz.
                                                This value
can be provided and adapted by the user application. */
#endif /* HSI_VALUE */

#if !defined (HSI48_VALUE)
#define HSI48_VALUE    ((uint32_t)48000000) /*!< Default value
of the HSI48 Internal oscillator in Hz.
                                                 This value
can be provided and adapted by the user application. */
#endif /* HSI48_VALUE */
/**
  * @}
  */

/** @addtogroup STM32F0xx_System_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F0xx_System_Private_Variables
  * @{
```

```
  */
  /* This variable is updated in three ways:
      1) by calling CMSIS function SystemCoreClockUpdate()
      2) by calling HAL API function HAL_RCC_GetHCLKFreq()
      3) each time HAL_RCC_ClockConfig() is called to
configure the system clock frequency
         Note: If you use this function to configure the
system clock there is no need to
               call the 2 first functions listed above, since
SystemCoreClock variable is
               updated automatically.
  */
uint32_t SystemCoreClock = 8000000;

const uint8_t AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1,
2, 3, 4, 6, 7, 8, 9};
const uint8_t APBPrescTable[8]  = {0, 0, 0, 0, 1, 2, 3, 4};

/**
  * @}
  */

/** @addtogroup STM32F0xx_System_Private_FunctionPrototypes
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F0xx_System_Private_Functions
  * @{
  */

/**
  * @brief  Setup the microcontroller system.
  *         Initialize the default HSI clock source, vector
table location and the PLL configuration is reset.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* Reset the RCC clock configuration to the default reset
state ------------*/
```

```
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001U;

#if defined (STM32F051x8) || defined (STM32F058x8)
  /* Reset SW[1:0], HPRE[3:0], PPRE[2:0], ADCPRE and MCOSEL[2:0] bits */
  RCC->CFGR &= (uint32_t)0xF8FFB80CU;
#else
  /* Reset SW[1:0], HPRE[3:0], PPRE[2:0], ADCPRE, MCOSEL[2:0], MCOPRE[2:0] and PLLNODIV bits */
  RCC->CFGR &= (uint32_t)0x08FFB80CU;
#endif /* STM32F051x8 or STM32F058x8 */

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFFU;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFFU;

  /* Reset PLLSRC, PLLXTPRE and PLLMUL[3:0] bits */
  RCC->CFGR &= (uint32_t)0xFFC0FFFFU;

  /* Reset PREDIV[3:0] bits */
  RCC->CFGR2 &= (uint32_t)0xFFFFFFF0U;

#if defined (STM32F072xB) || defined (STM32F078xx)
  /* Reset USART2SW[1:0], USART1SW[1:0], I2C1SW, CECSW, USBSW and ADCSW bits */
  RCC->CFGR3 &= (uint32_t)0xFFFCFE2CU;
#elif defined (STM32F071xB)
  /* Reset USART2SW[1:0], USART1SW[1:0], I2C1SW, CECSW and ADCSW bits */
  RCC->CFGR3 &= (uint32_t)0xFFFFCEACU;
#elif defined (STM32F091xC) || defined (STM32F098xx)
  /* Reset USART3SW[1:0], USART2SW[1:0], USART1SW[1:0], I2C1SW, CECSW and ADCSW bits */
  RCC->CFGR3 &= (uint32_t)0xFFF0FEACU;
#elif defined (STM32F030x6) || defined (STM32F030x8) || defined (STM32F031x6) || defined (STM32F038xx) || defined (STM32F030xC)
  /* Reset USART1SW[1:0], I2C1SW and ADCSW bits */
  RCC->CFGR3 &= (uint32_t)0xFFFFFEECU;
#elif defined (STM32F051x8) || defined (STM32F058xx)
  /* Reset USART1SW[1:0], I2C1SW, CECSW and ADCSW bits */
  RCC->CFGR3 &= (uint32_t)0xFFFFFEACU;
#elif defined (STM32F042x6) || defined (STM32F048xx)
```

```
  /* Reset USART1SW[1:0], I2C1SW, CECSW, USBSW and ADCSW bits
*/
  RCC->CFGR3 &= (uint32_t)0xFFFFFE2CU;
#elif defined (STM32F070x6) || defined (STM32F070xB)
  /* Reset USART1SW[1:0], I2C1SW, USBSW and ADCSW bits */
  RCC->CFGR3 &= (uint32_t)0xFFFFFE6CU;
  /* Set default USB clock to PLLCLK, since there is no HSI48
*/
  RCC->CFGR3 |= (uint32_t)0x00000080U;
#else
 #warning "No target selected"
#endif

  /* Reset HSI14 bit */
  RCC->CR2 &= (uint32_t)0xFFFFFFFEU;

  /* Disable all interrupts */
  RCC->CIR = 0x00000000U;

}

/**
   * @brief  Update SystemCoreClock variable according to
Clock Register Values.
  *         The SystemCoreClock variable contains the core
clock (HCLK), it can
  *         be used by the user application to setup the
SysTick timer or configure
  *         other parameters.
  *
  * @note   Each time the core clock (HCLK) changes, this
function must be called
  *         to update SystemCoreClock variable value.
Otherwise, any configuration
  *         based on this variable will be incorrect.
  *
  * @note   - The system frequency computed by this function
is not the real
  *           frequency in the chip. It is calculated based
on the predefined
  *           constant and the selected clock source:
  *
  *           - If SYSCLK source is HSI, SystemCoreClock will
contain the HSI_VALUE(*)
  *
  *           - If SYSCLK source is HSE, SystemCoreClock will
contain the HSE_VALUE(**)
  *
  *           - If SYSCLK source is PLL, SystemCoreClock will
contain the HSE_VALUE(**)
  *             or HSI_VALUE(*) multiplied/divided by the PLL
factors.
  *
  *         (*) HSI_VALUE is a constant defined in
stm32f0xx_hal.h file (default value
  *             8 MHz) but the real value may vary depending
on the variations
  *             in voltage and temperature.
  *
  *         (**) HSE_VALUE is a constant defined in
stm32f0xx_hal.h file (default value
  *              8 MHz), user has to ensure that HSE_VALUE is
same as the real
  *              frequency of the crystal used. Otherwise,
this function may
  *              have wrong result.
  *
  *         - The result of this function could be not
correct when using fractional
  *           value for HSE crystal.
  *
  * @param  None
  * @retval None
  */
void SystemCoreClockUpdate (void)
{
  uint32_t tmp = 0, pllmull = 0, pllsource = 0, predivfactor
= 0;

  /* Get SYSCLK source ---------------------------------------
-----------------*/
  tmp = RCC->CFGR & RCC_CFGR_SWS;

  switch (tmp)
  {
    case RCC_CFGR_SWS_HSI:  /* HSI used as system clock */
      SystemCoreClock = HSI_VALUE;
      break;
    case RCC_CFGR_SWS_HSE:  /* HSE used as system clock */
      SystemCoreClock = HSE_VALUE;
      break;
    case RCC_CFGR_SWS_PLL:  /* PLL used as system clock */
      /* Get PLL clock source and multiplication factor -----
-----------------*/
      pllmull = RCC->CFGR & RCC_CFGR_PLLMUL;
      pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;
      pllmull = ( pllmull >> 18) + 2;
      predivfactor = (RCC->CFGR2 & RCC_CFGR2_PREDIV) + 1;

      if (pllsource == RCC_CFGR_PLLSRC_HSE_PREDIV)
      {
        /* HSE used as PLL clock source : SystemCoreClock =
HSE/PREDIV * PLLMUL */
        SystemCoreClock = (HSE_VALUE/predivfactor) * pllmull;
      }
#if defined(STM32F042x6) || defined(STM32F048xx) ||
defined(STM32F072xB) || defined(STM32F078xx) ||
defined(STM32F091xC) || defined(STM32F098xx)
      else if (pllsource == RCC_CFGR_PLLSRC_HSI48_PREDIV)
      {
        /* HSI48 used as PLL clock source : SystemCoreClock =
HSI48/PREDIV * PLLMUL */
        SystemCoreClock = (HSI48_VALUE/predivfactor) *
pllmull;
      }
#endif /* STM32F042x6 || STM32F048xx || STM32F072xB ||
STM32F078xx || STM32F091xC || STM32F098xx */
      else
      {
#if defined(STM32F042x6) || defined(STM32F048xx)  ||
defined(STM32F070x6) \
 || defined(STM32F078xx) || defined(STM32F071xB)  ||
defined(STM32F072xB) \
 || defined(STM32F070xB) || defined(STM32F091xC) ||
defined(STM32F098xx)  || defined(STM32F030xC)
        /* HSI used as PLL clock source : SystemCoreClock =
HSI/PREDIV * PLLMUL */
        SystemCoreClock = (HSI_VALUE/predivfactor) * pllmull;
#else
        /* HSI used as PLL clock source : SystemCoreClock =
HSI/2 * PLLMUL */
        SystemCoreClock = (HSI_VALUE >> 1) * pllmull;
#endif /* STM32F042x6 || STM32F048xx || STM32F070x6 ||
          STM32F071xB || STM32F072xB || STM32F078xx ||
STM32F070xB ||
          STM32F091xC || STM32F098xx || STM32F030xC */
      }
      break;
    default: /* HSI used as system clock */
      SystemCoreClock = HSI_VALUE;
      break;
  }
  /* Compute HCLK clock frequency ----------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> 4)];
  /* HCLK clock frequency */
  SystemCoreClock >>= tmp;
}

/**
  * @}
  */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics
*****END OF FILE****/
```

PRINTING WITHOUT SPACES

It may not be immediately apparent to the new Forth programmer how to print without spaces between items.

Examples

In this example the programmer wants the constant “ref1” to be printed with the character “V” immediately following, ie “10V”. The common Forth print statement is used in Voltage1 Example Code below.

```
.” Hello”  ( - - ) Compiles a string and prints it when
executed.
```

Voltage1 Example Code

```
10 constant ref1

: voltage1 ( -- )  ref1 . ." V" cr ;
```

Voltage1 Output

```
voltage1  10 V
```

As you can see there is a space between the “10” and the “V” which is *not* what was wanted. The behaviour of this print method is the correct and desired behaviour, so what do we do ?

The Solution

The solution is to use Pictured Numerical Output which can format a text string however one desires.

Voltage2 Example Code

```
10 constant ref1

: voltage2 ( -- )  ref1    0 <# #s #> type  ." V"  cr ;
```

Voltage2 Output

```
voltage2  10V
```

Explanation

The word #S converts the value on the stack into ASCII characters. It will only produce as many digits as are necessary to represent the number; it will not produce leading zeroes. But it always produces at least one digit, which will be zero if the value was zero.

The Philosophy of Problems

A common issue new Mecrisp-Stellaris users encounter is the “Jump too far” compiler error message.

See also

compiler error messages.

Error Message	Description
Jump too far	All jumps use Thumb1 short opcodes. Solution; split your code into smaller Words

I fought thru this myself years ago and I found it *very annoying* because it was *stopping* my program from working, at least that was my initial reaction.

As tempting as it is to blame Mecrisp-Stellaris for this impediment, the real issues like most things in life are far more complex.

Please bear with me as I detour slightly.

Forth is above all else, a PROBLEM SOLVING SYSTEM

```
It has been said that one does not write a program in Forth
           Rather, one extends Forth to make a new language
                                      Specifically designed
                                For the application at hand
```

I think this is a very important principle, so important that I’m always asking myself, “what is the problem I’m trying to solve?”.

I believe that the answer is always built into the question, but to get the *right* answer, the *right* question must be asked. If I don’t understand the problem, I can’t ask the *right* questions.

… back to our topic.

What Problem is Mecrisp-Stellaris solving ?

Why did Matthias Koch write Mecrisp-Stellaris, what problem(s) was he trying to solve ?

I’m obviously NOT Matthias Koch but I do have some observations gathered since I began learning Mecrisp-Stellaris Forth in 2014. These are my opinions, and only Matthias knows the actual motivations underlying his design. I don’t believe they are a secret and I’m sure Matthias would be pleased to give them if asked.

What do I know about Dr Koch ?

- Is a scientist who thinks in MSP430 Assembly Language
- Thinks “outside the box” and is highly creative.
- Studies the natural world to learn from Mother Nature

- Loves efficiency and abhors waste and bloat
- Loves solving complex algorithms
- Designed Mecrisp for himself, not for you or I, but freely gave it to us believing it may be useful.
- Believes that if he can do anything, so can you or I. Matthias is an equal opportunity genius.

Can we get to the point please ?

Based on what I know about Matthias, I believe he created Mecrisp-* because he wanted a highly efficient, small, fast, self optimising, reliable Forth to test and showcase his capabilities and to use in his daily work and life interfacing with the real world.

How does this concern the “Jump too far” error ?

Could Matthias make this error go away ?

I’m sure he could, but *would* he, that’s the real question.

My personal opinion is that his current strategy provides the fastest/most efficient solution for his design time and he probably feels that if we see this as a problem, then we are free to change Mecrisp-Stellaris for ourselves because his sourcecode is freely available under the GPL license.

Comment on the “Jump too far” error message from Matthias Koch

> *I considered adding longer jumps at the beginning, but then I ran into the problem that I could not reuse the same code on M0 as on M3/M4. Optimisation of the control structures is a tricky piece, as control structures*

> *require backpatching of jump opcodes. How much memory should be reserved for the opcode without knowing the necessary range in advance ? As I had no urgent need for longer jump ranges, I just did it the easy way: All jumps use M0 short opcodes.*
>
> *It’s mostly a balance of my needs vs the implimentation development time*

Rationale Behind This Page

A new user will inevitably encounter many or all of the Mecrisp-Stellaris Error Messages and anything that stops our code from working can be very annoying.

I believe it pays to understand the aims and goals of the Mecrisp-Stellaris Architect when trying to understand what an error message means, what it is trying to tell us to help fix *our* problem.

Finally, I’ve been using Mecrisp-Stellaris since 2014 and although I have *thought* I’ve found dozens of bugs, I have *never* found a single actual bug.

All I have discovered again and again, is my own ignorance of Forth or insufficient logic in my design, fortunately both conditions have a cure, read, design, code and build as often as you can.

Mecrisp Stellaris Unofficial User Documentation

To read this UserDoc in nicely formatted html webpages go to: https://mecrisp-stellaris-folkdoc.sourceforge.io/

File Contents

tp-allpatch-stm32f103ra-2.5.3.patch

See: https://mecrisp-stellaris-folkdoc.sourceforge.io/patches.html#stm32f103-patch for full details.

f103-developer-image.tar.gz

STM32F103 BluePill ‘Developer’ image. It’s a bootable binary, requires a 128kB STM32F103C8 or STM32F103CB MCU and a 3.3v USB/serial dongle with CTS. See the README for full details: https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/f103-developer-image.c9ebc3e9d6.README.html

STM32F0-Discovery-TouchSensor-Demo.bin

This project supplies a binary file ready to run on a STM32F0 Discovery Board. It provides a fully configured Touch Sensor and Menu so that various user created touch sensors may be tested and calibrated.

Complete information

https://mecrisp-stellaris-folkdoc.sourceforge.io/touch-sensor.html

svd2forth-STM32L4x6.tar.gz

Mecrisp-Stellaris Memory Mapped Forth and GAS support prebuilt for the STM32L4x6 MCU

stm32f1xx-diagnostics-vx.bin

https://mecrisp-stellaris-folkdoc.sourceforge.io/stm32f1xx-diagnostics.html#stm32f103xx-diagnostics

This is the new version of STM32F103C8-DIAGNOSTICS.bin which was my first exploratory program to explore what actual MCU was in any of the infamous Blue Pill boards. The Blue Pill suffered from many issues but the most serious was the use of ‘fake’ and ‘compatible’ non genuine STM32F103 chips.

The new stm32f103xx-diagnostics contains lessons learned from the above and adds new features such as :-

- Attempts to tell if the MCU is a STM32F,CS32F or APM32F or a GB32F
- Simpler, cleaner Main Menu
- Extra Menu
- Generates a XML dump of all the data with suggested unique filename. This can then be uploaded to a database, used for your records etc. See the “FAQ” for details.
- Should work with the Maple Mini board

- Halts nicely with a error message if the second 64kB Flash block fails

Plus the old capabilities:

- Under 64kB in size so it can be loaded and run from any STM32F1xx class with 64kB Flash
- Bootable
- Creates a “Mecrisp STM32F10x Forth Serial Port, class 2/0, rev 2.00/2.00, ” virtual serial port device on your PC
- Dump the contents of the first 64kB Flash block
- Dump the contents of the second 64kB Flash block
- FAQ’s, Credits, and Liucense are in the “Extra” menu
- Serial can be connected to USART2 instead of USB.

Changelog

V1.5: Minor change, more robust menu, should only quit on “q” entry now.

V1.4: Changed PB9 to permanent LOW for Maple Mini, menu is run from boot, Version number is automatically displayed in the menu

V1.3: Changed GD32F103 ID string from 0x76F to 0x412

f103-diagnostics-image-blinky.fs

https://mecrisp-stellaris-folkdoc.sourceforge.io/bluepill-blinky.html#bluepill-blinky

Gentle introduction to Forth via a PC13 LED Blinky for use with STM32F103C8-DIAGNOSTICS.bin running on a generic Blue Pill Board

STM32F103C8-DIAGNOSTICS.bin

https://mecrisp-stellaris-folkdoc.sourceforge.io/stm32f103c8-diags.html#stm32f103c8-diags

Check for ‘hidden 64kB Flash’ and test it, read ID’s in BluePill Boards. Bootable binary, flash-n-go, *uses the USB connector as a virtual serial port* with your choice of serial terminal. Works on Windows, Linux, BSD etc. Simple menu interface.

bluepill-128kb-usb.tar.gz

Bootable Forth USB image. Memstats, ChipID, most Peripheral registers, stack sensitive red-prompt, can switch live between USB and USART2 with hardware handshaking. Details in the tarball readme.

mecrisp-unofficial-doc-8oct19.tar.gz

Complete copy of everything on this site. Updated infrequently due to file size, see date of file for last update.

MecrispStellarisUnofficial.epub

Automatic Epub conversion of the documents on this site, not everything translates properly. Updated with every content update.

Note

Copyrights: All Open Source, either GPL or FreeBSD as indicated.

Sigrok

Forth Use in Spacecraft

RTX2010 Radiationed Hardened MCU

DEVELOPER	MISSION DESCRIPTION
NASA	Advanced Composition Explorer. Solar and space exploration mission to study matter comprising energetic particles from the solar wind, the interplanetary medium, and other sources.
NASA	NEAR Shoemaker. To study the near-Earth asteroid Eros from close orbit over a period of a year.
NASA	TIMED (Thermosphere Ionosphere Mesosphere Energetics and Dynamics) to study the dynamics of the Mesosphere and Lower Thermosphere (MLT) portion of the Earth’s atmosphere.
European Space Agency	Philae lander that accompanied the Rosetta spacecraft until it separated to land on comet 67P/Churyumov–Gerasimenko, ten years and eight months after departing Earth.

2002

Reprinted from Forth Inc, (the home of Forth Legend [e.d.rather]) https://www.forth.com/resources/space-applications/ who reprinted it from the 14 April 2003 version of NASA staff's list of the uses of Forth in space applications. The list is archived at the Internet Archive WayBack Machine. The original URL was: http://forth.gsfc.nasa.gov/, where it was curated by NASA's James Rash.

DEVELOPER	MISSION DESCRIPTION
Baja Technology LLC	IMAGE Controller for the Extreme Ultraviolet (EUV) instrument, using Harris RTX2010 Forth microprocessor
Baja Technology LLC	Rosetta Controller for the Ion and Electron Sensor instrument, using Harris RTX2010 Forth microprocessor
Baja Technology LLC	Deep Impact Controller for the Avionics & Control (AOC) system, using Harris RTX2010 Forth microprocessor
Ball Aerospace	Advanced Composition Explorer (ACE) Star Scanner instrument & control software, using Harris RTX 2000 Forth microprocessor

DEVELOPER	MISSION DESCRIPTION
Ball Aerospace	Chandra X-ray Observatory Software to control the Science Instrument selection and focus mechanisms, using Harris RTX 2010 Forth microprocessor
Ball Aerospace	Chandra X-ray Observatory Software to operate precision CCD Aspect Camera, using Harris RTX 2010 Forth microprocessor
Ball Aerospace	Earth Observing System (EOS) Scanning mirror controller for the Thermal Imaging Radiometer (TIR), using Harris RTX 2010 Forth microprocessor
Ball Aerospace	Midcourse Sensor Experiment (MSX) – SDIO Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor
Ball Aerospace	Military aircraft Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor

DEVELOPER	**MISSION DESCRIPTION**
Ball Aerospace	Miniature Seeker Technology Integration (MSTI) Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor
Ball Aerospace	Near Earth Asteroid Rendezvous (NEAR) Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor
Ball Aerospace	SAGE (Stratospheric Aerosol and Gas Experiment) III CCD Controller Software, using Harris RTX 2010 Forth microprocessor
Ball Aerospace	Shuttle Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor
Ball Aerospace	Submillimeter Wave Astronomy Satellite (SWAS) Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor

DEVELOPER	MISSION DESCRIPTION
Ball Aerospace	X-ray Timing Explorer (XTE) Star Tracker instrument & control software, using Harris RTX 2000 Forth microprocessor
California Institute of Technology	Advance Composition Explorer (ACE) Cosmic Ray Isotope Spectrometer (CRIS) instrument controller hardware (RTX2010) and software in Forth
California Institute of Technology	Advance Composition Explorer (ACE) Solar IsotopeSpectrometer (SIS) instrument controller hardware (RTX2010) and software in Forth
FORTH, Inc.	USAF Space-Based Infrared System (SBIRS) Position encoder implemented with SwiftX cross compiler on Harris RTX-2010 Forth microprocessor
Innovatia Laboratories	NASA & USAF tropospheric research balloons Telemetry system using Innovatia MC1 microcontroller running Forth-83

DEVELOPER	MISSION DESCRIPTION
Johns Hopkins University Applied PhysicsLaboratory	USAF DMSP Magnetometer instrument control
JHU/APL USAF DMSP	Special Sensor Ultraviolet Spectrographic Imager (SSUSI), 5 missions over 15 years using Harris RTX 2010 Forth microprocessor
JHU/APL	Advanced Composition Explorer (ACE) satellite ULEIS (Ultra Low Energy Isotope Spectrometer), using embedded Harris RTX 2010 Forth microprocessor
JHU/APL	CASSINI spacecraft Magnetospheric Imager (MIMI) using Harris RTX 2010 Forth microprocessor
JHU/APL	HILAT Magnetometer instrument control
JHU/APL	Hopkins Ultraviolet Telescope (HUT) Instrument control, on-line data analysis, telemetry

DEVELOPER	**MISSION DESCRIPTION**
JHU/APL	MAGSAT Attitude Control Processor
JHU/APL	Polar Bear (Polar Beacon & Auroral Research Satellite) Magnetometer instrument control
JHU/APL	US Navy LONARS High precision navigation receiver using Loran-C Real time data processing & sensor control software
JHU/APL	Near Earth Asteroid Rendezvous (NEAR) Command & Telemetry Processor,Multi-Spectral Imaging System,Infrared Spectrograph and Magnetometer,X-ray/Gamma-ray Spectrometer, andLaser Rangefinder, all using the Harris RTX 2010RH Forth microprocessor
Los Alamos National Laboratory	Advanced Composition Explorer (ACE) satellite Solar wind data processing unit, using Harris RTX 2010 Forth microprocessor

DEVELOPER	MISSION DESCRIPTION
McDonnell Douglas	Shuttle – Electrophoresis Operations in Space (1986) Pharmaceutical purification process control software
McGill University	International Microgravity Laboratory (IML-1) (shuttle mission STS-42) Space Adaptation Syndrom Experiment management and data acquisition for four out of seven life science experiments
NASA/GSFC	Compatibility Test Van (CTV) Dual channel PCM simulator system software
NASA/GSFC	Compatibility Test Van (CTV): Landsat, HST, GRO, COBE, UARS, etc., Wallops, Data Evaluation Lab Programmable Data Formatter III system software
NASA/GSFC	CRAF/CASSINI MASSACQ Flight Instrument Test/Calibration System
NASA/GSFC	Marine Optical Buoy (MOBY) Command and data acquisition system written in Forth

DEVELOPER	MISSION DESCRIPTION
NASA/GSFC	Massively Parallel Processor (MPP) Image processing software
NASA/GSFC	NCC Mission Operations Support Area (MOSA) ITR (Intelligent Transmitter/Receiver) — Interface board for MOSA (formerly Mini-NOCC) Front End
NASA/GSFC	Shuttle Solar Backscatter Ultra Violet (SSBUV) ozone calibration instrument with eight successful shuttle bay flights Small Payload Accommodation Module (SPAM) interface controller software written in Forth
NASA/GSFC	Shuttle Solar Backscatter Ultra Violet (SSBUV) Instrument Interface Electronics Module (IIEM) software written in Forth
NASA/GSFC	Shuttle Ozone Limb Sounder Experiment (SOLSE) A “Hitchhiker Junior”, to fly summer ‘97 — instrument controller writtenin Forth

DEVELOPER	**MISSION DESCRIPTION**
NASA/GSFC	Limb Ozone Retrieval Experiment (LORE) A small independent instrument within SOLSE — instrument controller written in Forth
NASA/GSFC	Ultraviolet Imaging Telescope (UIT) on Space Shuttle Astro-2 mission Dedicated Experiment Processor (DEP) written in Forth
NASA/GSFC	Small Spacecraft Technology Initiative (SSTI) (Lewis Research Center/TRW) Goddard Electronics Module (GEM) data acquisition processor using Harris RTX-2010
NASA/GSFC	ACE (JHU/APL) Spacecraft Loads & Acoustic Monitor (SLAM) simultaneous data acquisition and downlink subsystem using Harris RTX-2010
NASA/GSFC	Wideband Transport Frame Formatter (WTFF) Virtual channel processor & system controller for the Advanced Orbiting Systems (AOS) Testbed using Harris RTX 2000 Forth microprocessor

DEVELOPER	MISSION DESCRIPTION
NASA/JSC	Shuttle, IML-1 Krug Life Sciences Microgravity Vestibular Investigations
NASA/JPL	Shuttle Imaging Radar (SIR-B) Command & control software
NASA	TOPEX Portion of Ground Support Equipment software
NSI	Globalstar, Thuraya, Iridium, ICO, Astrolink, various NASA missions including SWAS (Sub millimeter wave astronomy satellite), Space Shuttle Synthetic Aperture Radar (SAR), and Rosetta (comet probe) Microwave nearfield antenna testing of spacecraft antennas
Orbital Sciences Corporation	ORBCOMM Experimental Package OXP-1 on Pegasus Flight & experiment software
Orbital Sciences Corporation	ORBCOMM Experimental Package OXP-1 on Pegasus Flight & experiment software

DEVELOPER	MISSION DESCRIPTION
Orbital Sciences Corporation	(formerly Fairchild Space & Defense) USAF Multimission Modular Spacecraft (MMS) Multiplex Data Bus Monitor/Recorder test equipment using Harris RTX2000 microprocessor and firmware written in Forth
Orbital Sciences Corporation	ORBCOMM-X Telemetry, flight control support
Orbital Sciences Corporation	USAF APEX Data management and power control software
Orbital Sciences Corporation	SeaStar Sea WiFs data management and power control software
Orbital Sciences Corporation/Hercules	Pegasus Telemetry & flight control

DEVELOPER	MISSION DESCRIPTION
Radio Amateur Satellite Corp. (AMSAT-DL (Germany))	Integrated Housekeeping Unit (including navigation, attitude control, telemetry and telecommand tracking, payload control, power management, health & safety, and propulsion control) and Ground Support Equipment, all using the Forth-like language IPS (Interpreter for Process Structures). OSCAR 21 used the Harris RTX2000 microprocessor.
Swedish Space Corp. and JHU/APL	Freja Magnetometer experiment control
University of California	Galileo Magnetic Field Experiment instrument control
University of Wisconsin	Wisconsin Ultraviolet Photo-Polarimeter Experiment (WUPPE) Instrument control
US Navy	SPINSAT Portion of ground support equipment software

DEVELOPER	MISSION DESCRIPTION
USSR	Baikonur Cosmodrome Telemetry data analysis & retrieval workstations
Vertex Communications Corporation	Satellite earth station antenna systems 7210 Monopulse Antenna Control System on Motorola 68030 platform

STM32F103 MCU

This old but still very capable chip is used in various cheap Chinese boards and some decent boards by STM (discovery, Nucleo) and Olimex etc.

Warning

There are a lot of cheap Chinese boards for sale that don’t work because they use fake or bad chips. I always recommend using a trusted vendor, such as STM, Olimex, Seed Studio, Arrow.com etc if you want a STM32F103 Board. Arrow.com are a very reputable source of boards and chips.

Hardware Handshaking Image

Because this MCU is still so popular I have made up a special STM32F103 RA Forth image (mecrisp-stellaris-stm32f103-rts_cts.bin) with *hardware handshaking* using the Mecrisp-Stellaris 2.5.1 stm32f103-RA kernel.

```
MS RA 2.5.1 tpmod for STM32F103 by Matthias Koch
```

How does it work ?

When Source Code is uploaded to a MCU running Forth, every “Word” is compiled on the chip as it is received and this takes a finite amount of time depending on the complexity of the Word, the speed of the MCU and the speed of the serial Baud rate.

At higher Baud rates (such as 115200) if the serial data stream to the chip is too fast to be compiled, Forth will “lock up” and the board will need to be reset to get it working again.

To prevent this “lock up” we have a few strategies:

1. End of Line Delay: Implemented at the serial terminal, (usually about 200 milliseconds) stops the PC from uploading source code to the MCU for 200 ms after *every line of code*! This gives the on chip compiler time to digest longer more complex Words, however this EOL delay is applied after *every line* whether needed or not and will significantly slow down large Source Code uploads.

2. RTS/CTS hardware handshaking: This uses USART2 “control signals” named “RTS” and “CTS” to synchronise data transmissions between the PC and the MCU.

SIGNAL	INPUT	OUTPUT
RTS		YES
CTS	YES	

Note

Don’t short OUTPUTS to +V or 0v while testing. This is OK with INPUTS however.

At the MCU, RTS is asserted (tied LOW) as long as USART2 is ready to receive new data. When the USART2 receive register is full, RTS is de-asserted (raised HIGH), indicating that the transmission (from the PC) is expected to stop at the end of the current frame.

At the PC, the CTS line is connected to the MCU RTS line, and if the CTS line is held LOW, the PC is free to send data to the MCU via your serial terminal. However if the CTS line is raised HIGH the PC will stop sending data until it is LOW again.

This way no wasteful fixed length EOL delay is needed because the MCU will simply pause the Terminal sending data only when it needs to, making this method the fastest way to upload Source Code.

Note

This Image uses USART2 on the STM32F103 which is DIFFERENT to the normal USART1 used on this MCU by Mecrisp-Stellaris. The reason is that on this MCU the USART1 RTS control signal is used by the USB system and therefore not available. Connect a 3.3v/USB Dongle between your PC and the MCU board as shown below. Make sure the dongle has a CTS connection as well as the usual RX/TX pins and power.

RTS/CTS handshaking:

You don’t have to use the handshaking for the Kernel to work, but it will need a SLOW 200mS “End Of Line Delay” for your serial terminal if you don’t. No EOL delay is needed with hardware handshaking as the MCU will tell the PC to stop uploading source code as required while compiling.

Baud Rate is 115200 Baud.

Gpio	MCU Signal	3.3v/USB Dongle
PA1	RTS	CTS
PA2	TX	RX
PA3	RX	TX

Note

Don’t forget to activate your serial terminal RTS/CTS hardware handshaking mode.

Picocom serial terminal hardware handshaking example

Change the “/dev/cuaU0” to suit your (Unix) machine.

```
picocom -b 115200 /dev/cuaU0  --omap delbs,crlf --flow h --send-cmd "ascii-xfr -s -l200" &
```

Other Kernel Mods

- Added CR to the standard LF so you don’t have to do that in your terminal to prevent “stair stepping”

DOWNLOADS

64KB Flash Image

This modified but base image runs fine in a 64KB Flash mcu.

Image (20.5KB)

MD5SUM

128KB Developers Image

This 128KB Developers Image (b6feae9831fcabcad8f45442f98d8c32) is preloaded with tons of development support, see below for details. It uses the 64KB Kernel above, so it includes RTS/CTS handshaking on the same GPIO pins etc,

Download it here (44KB): https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/STM32F103-developers-image.tar.gz/download

Kernel Properties

- Size: 99184 Bytes
- Requires 128KB Flash version of the STM32F103, ie Olimex-P103, Blue Pill etc.
- Clock is internal RC 8MHz

b6feae9831fcabcad8f45442f98d8c32 Image Contains

- every single STM32F103 register memory map and pretty print with legends
- Arm Core registers STK. and SCB. with memory map and pretty print with legends

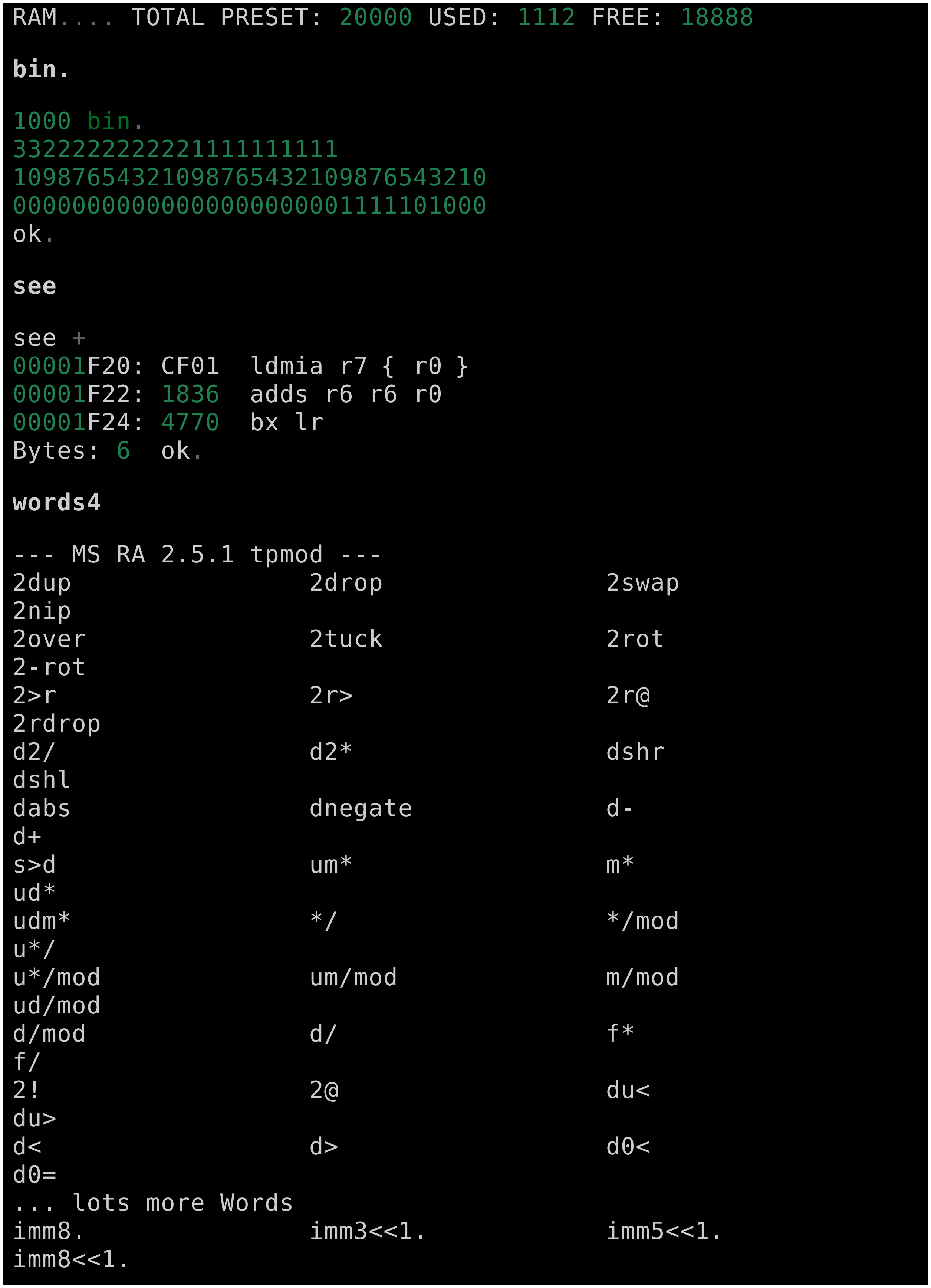
Note

See b6feae9831fcabcad8f45442f98d8c32.README for Specific Notes regarding Mecrisp-Stellaris Kernel b6feae9831fcabcad8f45442f98d8c32.bin and see b6feae9831fcabcad8f45442f98d8c32.words4.txt for the complete word list.

Words

free

```
free (bytes)
FLASH.. TOTAL REPORTED: 131072 USED: 99176 FREE: 31896
```



```
RAM.... TOTAL PRESET: 20000 USED: 1112 FREE: 18888

bin.

1000 bin.
3322222222221111111111
10987654321098765432109876543210
00000000000000000000001111101000
ok.

see

see +
00001F20: CF01  ldmia r7 { r0 }
00001F22: 1836  adds r6 r6 r0
00001F24: 4770  bx lr
Bytes: 6  ok.

words4

--- MS RA 2.5.1 tpmod ---
2dup                2drop               2swap
2nip
2over               2tuck               2rot
2-rot
2>r                 2r>                 2r@
2rdrop
d2/                 d2*                 dshr
dshl
dabs                dnegate             d-
d+
s>d                 um*                 m*
ud*
udm*                */                  */mod
u*/
u*/mod              um/mod              m/mod
ud/mod
d/mod               d/                  f*
f/
2!                  2@                  du<
du>
d<                  d>                  d0<
d0=
... lots more Words
imm8.               imm3<<1.            imm5<<1.
imm8<<1.
```

```
imm3<<2.            imm5<<2.            imm7<<2.
imm8<<2.
destination-r0      disasm              memstamp
disasm-step
seec                see                 pr
ok.
```

ansi colors

For use with the terminal: black, red, green, yellow, blue, magenta, cyan, white

Every single STM32F103 peripheral

Register can be pretty printed in real time with legends.

PWR.
RCC.
GPIOA.
GPIOB.
GPIOC.
GPIOD.
AFIO.
EXTI.
DMA1.
DMA2.
SDIO.
RTC.
BKP.
IWDG.
WWDG.
TIM1.
TIM2.
TIM3.
TIM4.
I2C1.
I2C2.
SPI1.
SPI2.
USART1.
USART2.
USART3.
ADC1.
CAN.
DBG.
CRC.

```
FLASH.
NVIC.
USB.
```

Example 1:

```
gpioa.
GPIOA_CRL (read-write) $44444AA4
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0
0

GPIOA_CRH (read-write) $88844444
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0

GPIOA_IDR (read-only) $00001F78
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0
0

GPIOA_ODR (read-write) $0000A000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_BSRR (write-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_BRR (write-only) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
```

```
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

GPIOA_LCKR (read-write) $00000000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 ok.
```

Example 2:

```
GPIOA_ODR.
GPIOA_ODR (read-write) $0000A000
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0
 ok.
```

System Control Block

Pretty print with legends in real time.

```
scb.
SCB_CPUID () $411FC231
SCB_CPUID BASE REGISTER
                       |  CONS- |
--IMPLEMENTER--| VARIANT| -TANT--| ------PART NUMBER------| RE-
3| 3| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|            | -
VISION
1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0
1

SCB_ICSR () $00000000
INTERRUPT CONTROL AND STATE REGISTER
Provides:
*A set-pending bit for the Non-Maskable Interrupt NMI
exception
*Set-pending and clear-pending bits for the PendSV and
SysTick
```

```
exceptions.
Indicates:
*The exception number of the exception being processed
*Whether there are preempted active exceptions
*The exception number of the highest priority pending
exception
*Whether any interrupts are pending.
N                 I
M     P P P P     S
I     E E E E     R
P     N N N N     P
E     D D D D     E
N     S S S S     N
D     V V T T     D
S     S C S C     I
E     E L E L     N
T     T R T R     G        |------VECT PENDING-----|VECT
ACTIVE
3|   |2|2|2|2|   |2|       |1|1|1|1|1|1|           |----------
--
1|~~~|8|7|6|5|~~~|2|~~~~~~~|7|6|5|4|3|2|~~~~~~~~~~~|5|4|3|2|1
|0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

SCB_AIRCR () $FA050000
APPLICATION INTERRUPT AND RESET CONTROL REGISTER            V
provides endian status for data accesses and reset         E
control of the system.                                    S C
                                E                         Y T
                                N                         S C
                                D                         R L
                                I                         E R
To write to this register, you  A                         S A
must write $5FA to the VECTKEY  N                         E C
field FIRST.                    E                         T T
                                S                         R I
-----------VECTKEY-------------|S|                        E V
3|3|2|2|2|2|2|2|2|2|2|2|1|1|1|1|1|                        Q E
1|0|9|8|7|6|5|4|3|2|1|0|9|8|7|6|5|~~~~~~~~~~~~~~~~~~~~~~~|2|1
|~
1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

SCB_SCR () $00000000
```

```
SYSTEM CONTROL REGISTER                                   S S
                                                      S   L E
controls features of entry to and exit from           E   E E
LOW POWER state.                                      V   E P
                                                      O   P O
                                                      N     N
                                                      P   D E
                                                      E   E X
                                                      N   E I
                                                      D   P T
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|4|~|2|1
|~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

SCB_CCR () $00000000
CONFIGURATION AND CONTROL REGISTER                      U
                                            S           N
read-only register and indicates some       T           A
aspects of the behavior of the              K           L
Cortex-M0 processor.                                    I
                                            A           G
                                            L           N
                                            I           T
                                            G           R
                                           |N           P
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|9|~~~~~~~~~|3|~~~
~~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

SCB_SHPR2 () $00000000
System handler priority registers (SHPRx)
The SHPR2-SHPR3 registers set the priority level, 0 to 192,
of the exception handlers that have configurable priority.
SHPR2-SHPR3 are word accessible.
----PRI_11-----|
3|3|2|2|2|2|2|2|
1|0|9|8|7|6|5|4|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

SCB_SHPR3 () $00000000
System handler priority registers (SHPRx)
```

```
The SHPR2-SHPR3 registers set the priority level, 0 to 192,
of the exception handlers that have configurable priority.
SHPR2-SHPR3 are word accessible.
----PRI_11-----|
3| 3| 2| 2| 2| 2| 2| 2|
1| 0| 9| 8| 7| 6| 5| 4| ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
ok.
```

Systick peripheral

Pretty print with legends in real time.

```
stk.
STK_CSR () $00000000
SysTick control and status register
The SysTick CSR register enables the SysTick features.
                              C
                              O
                              U                           C
                              N                           K T
                              T                           S I
E
                              F                           O C
N
                              L                           U K
A
                              A                           R I
B
                              G                           C N
L
                             |1|                          E
T| E
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|6| ~~~~~~~~~~~~~~~~~~~~~~~~~|2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

STK_RVR () $00000000
SysTick reload value register
The STK_RVR specifies the start value to load into the
STK_CVR.
               |--------------------RELOAD-------------------
```

```
--
               | 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
~~~~~~~~~~~~~~~| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

STK_CVR () $00000000









               |--------------------CURRENT------------------
--
               | 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
~~~~~~~~~~~~~~~| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0

STK_CALIB () $40002328
SysTick calibration value register
If calibration information is not known, calculate the
calibration value from the clock.
N
O S
R K
E E
F W            |---------------------TENMS-------------------
--
3| 3|          | 2| 2| 2| 2| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 0| ~~~~~~~~~~| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0| 9| 8| 7| 6| 5| 4| 3| 2| 1
| 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0
0

 ok.
```

72 MHz Clock Word

This Word utilizes the 8Mhz Crystal on all the popular boards and the MCU internal PLL to boost the clock to 72 MHz, which is nine times faster than the standard Mecrisp-Stellaris image.

This standalone Word requires no other support and uses absolute addressing to be as small as possible. It is designed to be loaded first to speed up all subsequent operations. As a result the source isn’t easy to grok, but at least it’s well commented.

“72mhz” is configured for 115200 Baud and USART1, which is the Mecrisp-Stellaris default. Other speed and USART options are available but commented out.

It will work with any STM32F103 Kernel including my recent Special STM32F103 image with RTS/CTS hardware handshaking and 128KB Flash STM32F103 image with RTS/CTS hardware handshaking images with some option changes such as USART and Baud rate.

72Mhz Sourcecode

```
\ Copyright 2019  t.porter <terry@tjporter.com.au>, licensed under the GPL V2
\ For STM32F103 Mecrisp-Stellaris by Matthias Koch

: 72mhz ( -- )  cr            \ Increase 8Mhz RC clock to 72 MHz via 8MHz Xtal and PLL.
  $12 $40022000 !             \ two flash mem wait states
  1 16 lshift  $40021000 bis! \ set HSEON
  begin 1 17 lshift $40021000
       bit@ until             \ wait for HSERDY
  1 16 lshift                 \ HSE clock is 8 MHz Xtal source for PLL
  7 18 lshift or              \ PLL factor: 8 MHz * 9 = 72 MHz = HCLK
  4  8 lshift or              \ PCLK1 = HCLK/2 = 36MHz
  2 14 lshift or              \ ADCPRE = PCLK2/6
  2 or $40021004  !           \ PLL is the system clock
  1 24 lshift $40021000 bis!  \ set PLLON
  begin 1 25 lshift $40021000
       bit@ until             \ wait for PLLRDY

  312 $40013808 !  \ USART1 @ 115200 Baud
  \ Options:
  \ 78  $40013808 ! \ USART1 @  460800 Baud.
  \ 312 $40004408 !  \ USART2 @ 115200 Baud
  \ 78  $40004408 !   \ USART2 @ 460800 Baud.

  ." Clock is now 72 MHz. " cr
;

```

Download 72mhz-f103.fs

Video

There is also a realtime video showing how fast the [128KB Developers Image]() lists all the Words on my Forth development system when the board is running at 72MHz. See: https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/stm32f103-devkernel-wordlist%4072mhz.mp4/download

This video is also on Youtube at: https://www.youtube.com/watch?v=JuW_noOexng&feature=youtu.be

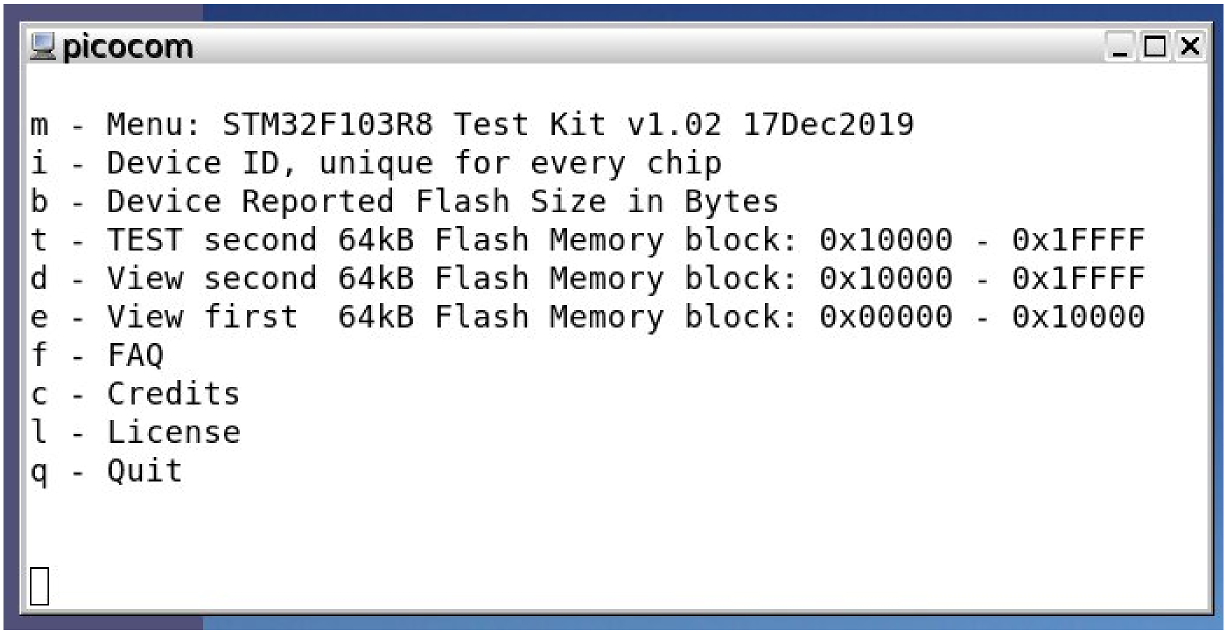
Summary

- Hardware Handshaking
- Gnu Screen terminal
- Baud Rate: 460800
- Clock is 72 MHz

STM32F103C8T6, 2 x 64kB flash ?

Technically, only the

STM32F103C8 Diagnostics



```
m - Menu: STM32F103R8 Test Kit v1.02 17Dec2019
i - Device ID, unique for every chip
b - Device Reported Flash Size in Bytes
t - TEST second 64kB Flash Memory block: 0x10000 - 0x1FFFF
d - View second 64kB Flash Memory block: 0x10000 - 0x1FFFF
e - View first  64kB Flash Memory block: 0x00000 - 0x10000
f - FAQ
c - Credits
l - License
q - Quit
```

STM32F103C8 datasheets specify the on board Flash memory as being 64kB, and this is corroborated by the on chip “Flash Size Register” having a value of 65536 Bytes, however it seems these chips actually have a “hidden” second 64kB Block, making a total of **128kB of Flash onboard!**

Many theories abound, did STM simply use the same die for both the more expensive 128kB Flash STM32F103CB and the cheaper STM32F103C8 to save costs, but configure the STM32F103C8 to show only 64kB ?

Perhaps STM ran out of STM32F103C8’s and simply used modified STM32F103CB units instead ?

Or maybe STM32F103CB units with a faulty memory location in the second Flash block were sold as

STM32F103C8’s and the”Flash Size Register” set to 65536 Bytes ?

The STM32F103C8 MCU is found in many cheap Chinese boards including the very popular Bluepill Board which means that many owners may not be aware they actually have *DOUBLE the Flash* they thought they had.

This Project

This is a bootable diagnostic program test kit. The size is under 64kB so that it can be flashed by OpenOCD.

The purpose of this kit is to physically test the “hidden” 64kB second Flash block present on many of these chips and determine if it exists and if it is reliable. It does this by flashing all the bits from one to zero and back again.

Download the *STM32F103C8-DIAGNOSTICS.bin* here: https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/STM32F103C8-DIAGNOSTICS.bin/download

Usage

1. Flash the STM32F103C8-DIAGNOSTICS.bin image to your board with OpenOCD or whatever you normally use. See Flashing Methods, but basically you flash this binary exactly as you would flash any STM32Fxx binary. .i.e a. “st-flash erase” b. “st-flash write STM32F103C8-DIAGNOSTICS.bin 0x08000000”
2. Plug in a USB cable to the “Blue Pill” board and run a serial terminal program on your PC. This will also power the board.
3. Connect the terminal to the ‘“Mecrisp STM32F10x Forth Serial Port’ USB device now created on your PC. Any

speed is ok, I use 460800 Baud myself.

4. Enter “?” at the prompt to obtain the menu.
5. The “t” key runs the all important Flash test, but there are other menu functions available as listed below.

Note

The first few versions had a garbled USB device ID string, this was corrected in V1.2 and should look something like this.

```
umodem1: <Mecrisp STM32F10x Forth Serial Port, class 2/0, rev
2.00/2.00, addr 15> on usbus0
```

Menu

Enter ? for the menu

```
m - Menu: STM32F103R8 Test Kit v1.2 24Dec2019
i - Device ID, unique for every chip
b - Device Reported Flash Size in Bytes
t - TEST second 64kB Flash Memory block: 0x10000 - 0x1FFFF
d - View second 64kB Flash Memory block: 0x10000 - 0x1FFFF
e - View first  64kB Flash Memory block: 0x00000 - 0x10000
f - FAQ
c - Credits
l - License
q - Quit
```

i

```
stm32id
Die xy coords: 108003155
Wafer Number: 80
Lot_num ascii encoded  [23:0]: 0x00555071  |  U P q
Lot_num ascii encoded [55:24]: 0x87042957  |  . . ) W
```

b

```
Flash Size Register = 65536 Bytes
```

t

This tests every bit of the *‘hidden’* flash block at 0x10000 - 0x1FFFF by writing alternating ones and zero’s, checking they have been saved and then erasing them. If you don’t see this result, or see error messages instead, then the *‘hidden’* flash block at 0x10000 - 0x1FFFF has a problem and probably is not available for use.

A successful test looks like this. Note: there are only three Block erasures and two block flashes when this test is run, so the Flash wear is negligible.

```
Testing 64kB Flash block: 0x10000 - 0x1FFFF
Erasing
Flash with 1010101010101010 (0xAA)
Testing for 0xAA - OK
Erasing
Flash with 0101010101010101 (0x55)
Testing for 0x55 - OK
Erasing
~~~~~ ALL TESTS PASSED ~~~~~
```

If test *“b”* shows 65536 Bytes yet this test passes as above, then you have a “hidden” extra 64kB of Flash which cannot be flashed by OpenOCD unless it is modified.

Note

If you try to flash a binary image of OVER 64kB into a STM32F103C8 for which the MCU *“Flash Size Register”* = 65536 Bytes, a standard OpenOCD will ONLY flash the first 64kB of your image. See below for how to utilise the full 128kB.

Sample OpenOCD modification to allow flashing the “hidden” extra 64kB of Flash

stm32f1x.cfg Modification

```
# flash size will be probed
set _FLASHNAME $_CHIPNAME.flash
# flash bank $_FLASHNAME stm32f1x 0x08000000 0 0 0
$_TARGETNAME        <---- comment this line out, add the line
below:
flash bank $_FLASHNAME stm32f1x 0x08000000 0x20000 0 0
$_TARGETNAME
```

Note

Forth doesn’t care about the Flash Size Register value and will use all 128kB of flash in this case.

d

This dumps the entire 64kB Flash block: 0x10000 - 0x1FFFF showing the binary and ascii contents.

e

This dumps the entire 64kB Flash block: 0x00000 - 0x10000 showing the binary and ascii contents.

f

- Will this work with non STM32F103R8 chips << Not guaranteed
- How many times can the ‘t’ test be safely run << 1000 times at least
- The Flash Data View is too fast << Use your terminal loging facility & capture it

- Will this test harm my chip << No
- How can I view the current chip memory status << Quit the Menu and enter ‘free’
- What was the Test Kit made with << The Forth Programming Language
- How do I find other programs I can run << Quit the Menu and enter ‘words4’. (Some of these Words can erase the Flash)
- Can I write other programs, blink a LED << Yes, quit the Menu and enter ‘gpioc.’ (This is a demo of the built in register read/write facility.)
- Learn more << https://mecrisp-stellaris-folkdoc.sourceforge.io/quickstart.html

c

Credits

```
This diagnostic program written by Terry Porter
<terry@tjporter.com.au>
https://mecrisp-stellaris-folkdoc.sourceforge.io/stm32f103c8-
diags.html
Get Mecrisp-Stellaris Forth: https://mecrisp-stellaris-
folkdoc.sourceforge.io
Mecrisp-Stellaris created by Matthias Koch
```

l

License

```
Mecrisp-Stellaris is free software: you can redistribute it
and/or modify
it under the terms of the GNU General Public License as
published by
the Free Software Foundation, either version 3 of the
License, or
(at your option) any later version.

This program is distributed in the hope that it will be
```

```
useful,
but WITHOUT ANY WARRANTY; without even the implied warranty
of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public
License
along with this program.  If not, see
<http://www.gnu.org/licenses/>.
```

q

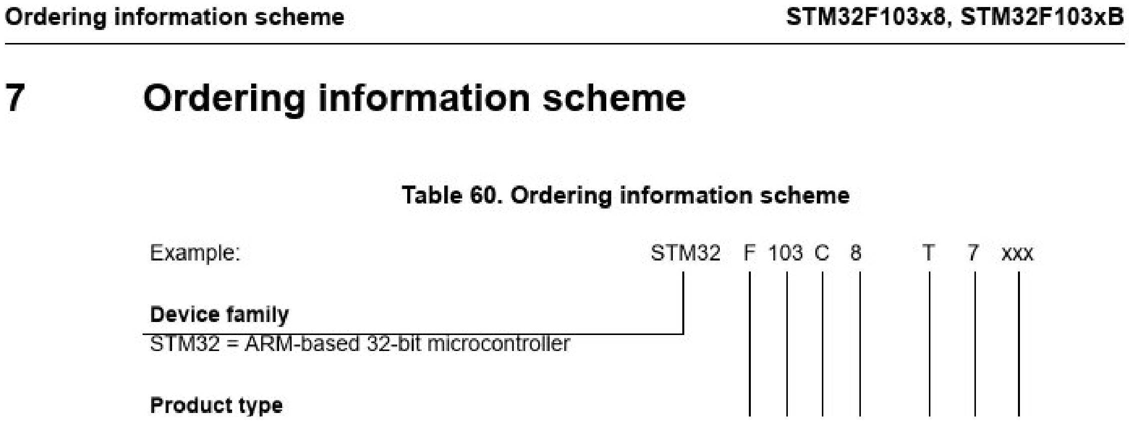
Quit the menu.

Warning

This is a full Cortex-M3 Forth and can be used from the terminal command line. If your chip fails test *"t"* don't use more than 64kB of your flash with the on-board Forth or it will crash.

Developed on a STM32F103C8 board (with the "hidden" second 64kB Flash block present) which is identical to a "Blue Pill", so it should work on all such boards.

STM32F103 Spec Sheet



Ordering information scheme STM32F103x8, STM32F103xB

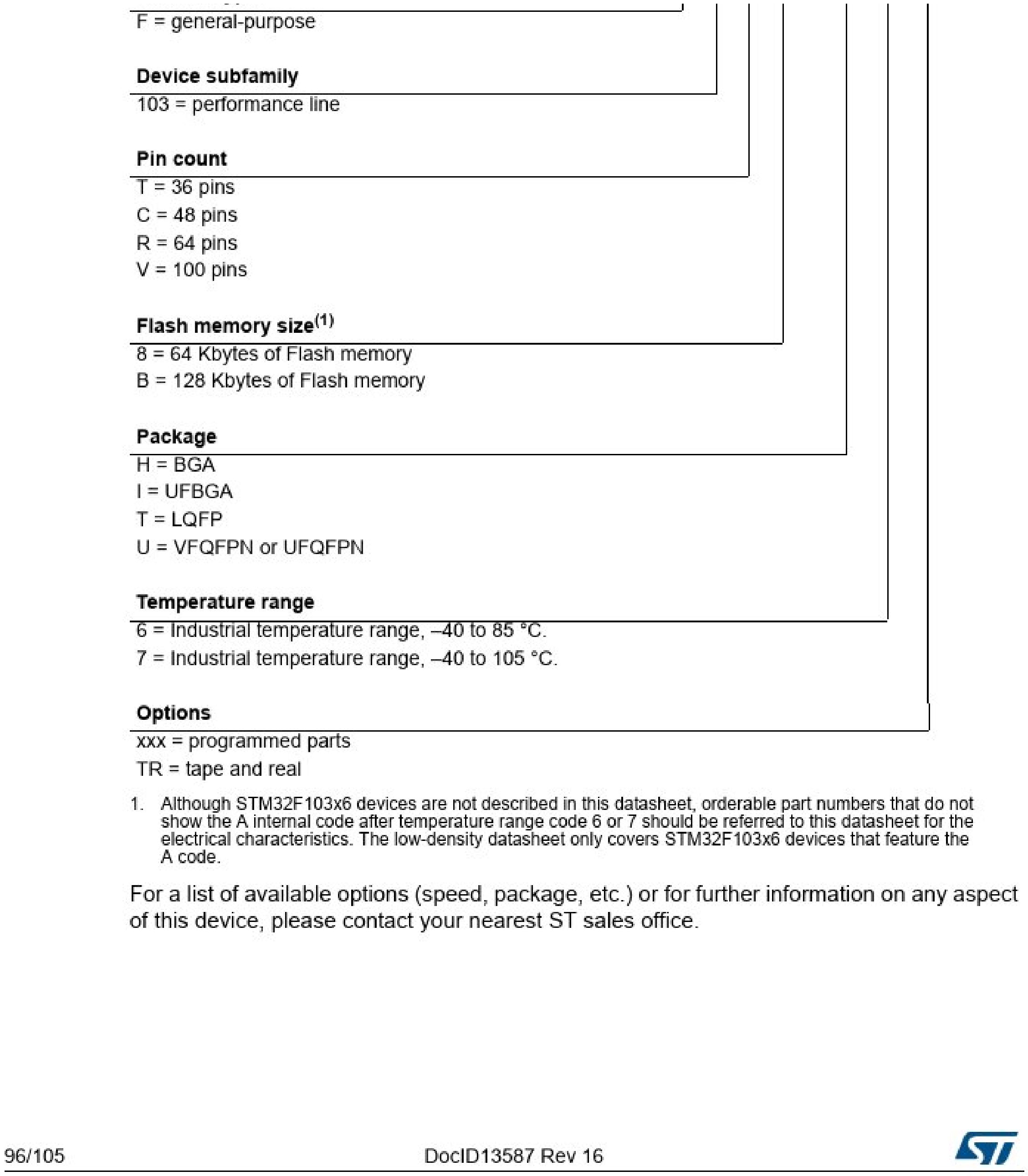
7 Ordering information scheme

Table 60. Ordering information scheme

Example: STM32 F 103 C 8 T 7 xxx

Device family
STM32 = ARM-based 32-bit microcontroller

Product type



F = general-purpose

Device subfamily
103 = performance line

Pin count
T = 36 pins
C = 48 pins
R = 64 pins
V = 100 pins

Flash memory size(1)
8 = 64 Kbytes of Flash memory
B = 128 Kbytes of Flash memory

Package
H = BGA
I = UFBGA
T = LQFP
U = VFQFPN or UFQFPN

Temperature range
6 = Industrial temperature range, –40 to 85 °C.
7 = Industrial temperature range, –40 to 105 °C.

Options
xxx = programmed parts
TR = tape and real

1. Although STM32F103x6 devices are not described in this datasheet, orderable part numbers that do not show the A internal code after temperature range code 6 or 7 should be referred to this datasheet for the electrical characteristics. The low-density datasheet only covers STM32F103x6 devices that feature the A code.

For a list of available options (speed, package, etc.) or for further information on any aspect of this device, please contact your nearest ST sales office.

96/105 DocID13587 Rev 16

How does this all work?

The fact that my 10 tested STM32F103C8’s had a second hidden 64kB block doesn’t alter the STM spec above because any flashing software must check the MCU internal “Flash Size Register” first, will read that the STM32F103C8 has only 64kB and ignore that second block, it doesn’t have any other choice.

Reading the MCU “Flash Size Register” first is essential for a few reasons one of which is the MCU will throw an exception if a program attempts to do anything to Flash that’s not actually there.

My St-Link would program any size image up to 64kB on the STM32F103C8. If the Image was 100kB, it stopped at 64kB and threw an error as expected.

St-Link reads the MCU “Flash Size Register” and acts accordingly as it must. It can be reconfigured so it thinks the STM32F103C8 has 128kB of Flash, which is what I had to do to flash binaries over 64kB but under 128kB on a STM32F103C8 which actually had the ‘hidden second 64kB Flash Block”.

My Olimex-P103 has a STM32F103CB with 128kB as shown in the Factory Datasheet above, so St-Link would program a 100kB image without complaint because the STM32F103CB MCU “Flash Size Register” indicates the chip has 128kB of Flash.

This is how it all works This “STM32F103C8 Diagnostics” bootable binary is designed to run on any STM32F103Cx as it is under 64kB and reveal all the above.

Chip ID Data

Can we tell from the chip ID data if a chip is a fake ?

I think so as genuine STM chip Lot Numbers are in DECIMAL, CS32F103C8T6 chip Lot Numbers are in Hexadecimal.

Chip	Lot Numbers	Example	Description
Genuine STM	decimal	0x00565283	Only contains digits 0 - 9
CS32F103C8T6	hexadecimal	0x004D3512	Contains digits 0 - 9 AND characters A,B,C,D,E,F
GD32F103C8T6	?		I have no data yet

Good Chip Reports

107741012/72

```
stm32id
Die xy coords: 107741012
Wafer Number: 72
Lot_num ascii encoded  [23:0]: 0x00565283  |  V R .
Lot_num ascii encoded [55:24]: 0x87242040  |  . $   @

Flash Size Register = 65536 Bytes
~~~~~ ALL TESTS PASSED ~~~~~
```

107872085/72

```
stm32id
Die xy coords: 107872085
Wafer Number: 72
```

```
Lot_num ascii encoded  [23:0]: 0x00555787  |  U W .
Lot_num ascii encoded [55:24]: 0x87061320  |  . . .

Flash Size Register = 65536 Bytes
~~~~~ ALL TESTS PASSED ~~~~~
```

Bad Chip Reports

CKS CS32F103C8T6

Rhodges from EEVblog reports a CKS CS32F103C8T6 that an ebay seller sold as a ST part (oops!) failed the test with 0x10000 - 0x1FFFF appearing to be a read-only mirror of 0x00000 - 0x10000

```
stm32id
Die xy coords: 16917299
Wafer Number: 50
Lot_num ascii encoded  [23:0]: 0x004D3712  |  M 7 .
Lot_num ascii encoded [55:24]: 0x004E4B43  |  . N K C

Flash Size Register = 65536 Bytes

Testing 64kB Flash block: 0x10000 - 0x1FFFF
Erasing
FAILTEST: $10000  Data = $8C
FAILTEST: $10001  Data = $03
... all the way to
FAILTEST: $1CDE5  Data = $BD
FAILTEST: $1CDE6  Data = $00
FAILTEST: $1CDE7  Data = $00
blank check FAILTEST
```

STM32F103XX-DIAGNOSTICS

This is the new version of STM32F103C8-DIAGNOSTICS.bin which was my first exploratory program to explore which actual MCU was in any of the infamous Blue Pill boards. STM32F103C8-DIAGNOSTICS.bin has been downloaded 190+ times so far.

The Blue Pill suffers from many issues but the most serious is the use of ‘fake’ and ‘compatible’ non genuine STM32F103 chips because they don’t always work the same as the genuine ones.

The new stm32f103xx-diagnostics contains lessons learned from the above and adds new features such as :-

- Attempts to tell if the MCU is a STM32F, CS32F or APM32F or a GD32F
- Simpler, cleaner Main Menu
- Extra Menu
- Generates a XML dump of all the data with suggested unique filename. This can them be uploaded to a database to examine what chips and memory have been found. This can then be uploaded, see the “FAQ” for details.
- Should work with the Maple Mini board
- Halts nicely with a error message if the second 64kB Flash block fails

Plus the old capabilities:

- Under 64kB in size so it can be loaded and run from any STM32F1xx class with 64kB Flash
- Bootable
- Creates a “Mecrisp STM32F10x Forth Serial Port, class 2/0, rev 2.00/2.00, ” virtual serial port device on your PC
- Dump the contents of the first 64kB Flash block
- Dump the contents of the second 64kB Flash block
- FAQ’s, Credits, and Liucense are in the “Extra” menu
- Serial can be connected to USART2 instead of USB.

Operation

1. Flash the stm32fxx-diagnostics-vx.bin image to your board with OpenOCD or whatever you normally use. See Flashing Methods, but basically you flash this binary exactly as you would flash any STM32Fxx binary. .i.e
    a. “st-flash erase”
    b. “st-flash write stm32fxx-diagnostics-vx.bin 0x08000000”

Warning

After flashing, disconnect any JTAG or SWD hardware and reboot the board or the incorrect “Debug” data will be collected.

2. Plug in a USB cable to the “Blue Pill/maple mini” board and run a serial terminal program on your PC. This will also power the board.
3. Connect the terminal to the ‘“Mecrisp STM32F10x Forth Serial Port’ USB device now created on your PC. Any speed is ok, I use 460800 Baud myself.
4. Enter “?” at the prompt to obtain the menu.

    a. Press the “t” key, this will test any “hidden” second 64kB block.
    b. Press “s” key and select which chip marking best describe the chip in your board
5. The Summary will be updated at all times
6. Visit the “extra” menu and upload your XML FILE (not a picture please) to the URL or email it to me at terry@tjporter.com.au” for me to fed into the database. The results will be displayed here when I have enough data.

Board not supported

If you have a board which is sold in large numbers and it doesn’t work with the USB connection as above I’m happy to add the capability to this test program if I can.

To do so, I need to know which GPIO is connected to the USB “USBDP” signal and the ‘normal’ and ‘pulsed’ state. See the USB picture above for more details or email me. Failing that I need a PDF or picture of your schematic showing these connections which can also be emailed to me.

DOWNLOAD HERE

https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/stm32fxx-diagnostics-v1.4.bin/download

Changelog

V1.4: Changed PB9 to permanent LOW for Maple Mini, menu is run from boot, Version number is automatically displayed in the menu

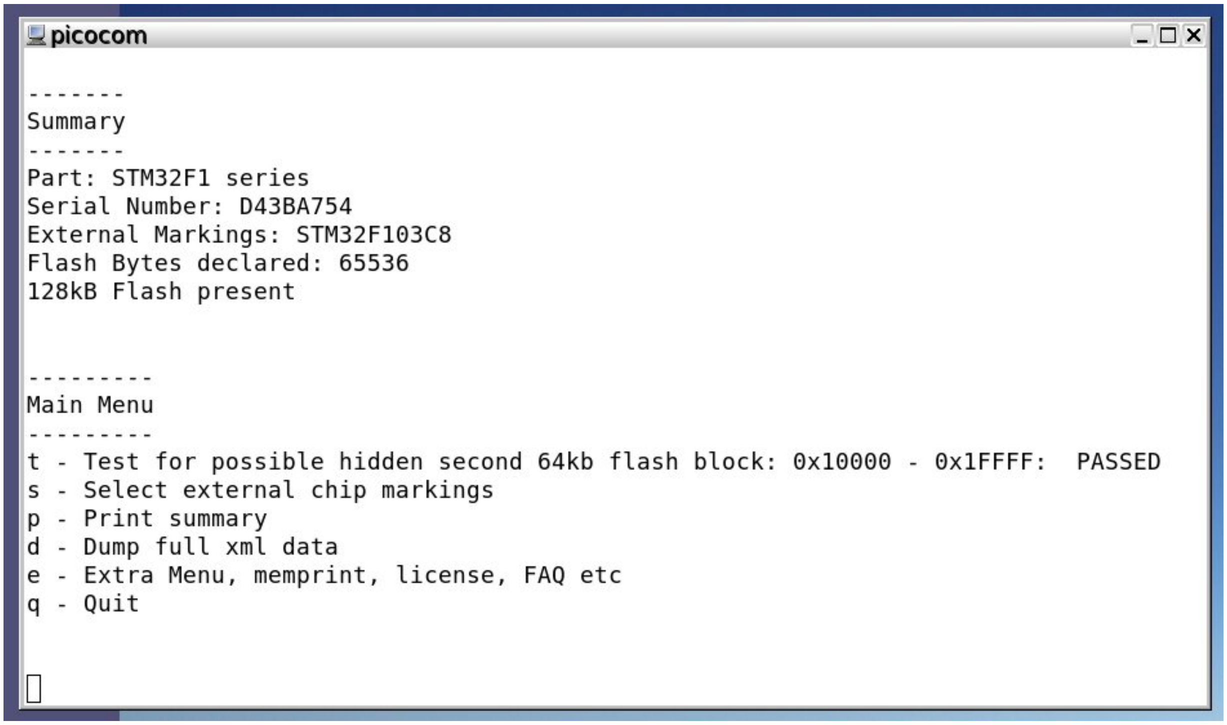
Caveat

This is V1.0 so it may not work for you or have bugs. Please email me with details if you find any and I'll fix them if I can and release a new version. I only have STM32F103C8 and STM32F103CB MCU's. I don't have any GD32F103, CS32F103 or APM32F103 chips so I'm unable to test them.

If you want to send me any boards with the above chips, just email and I'll supply a Australian address for you to ship to me for testing.

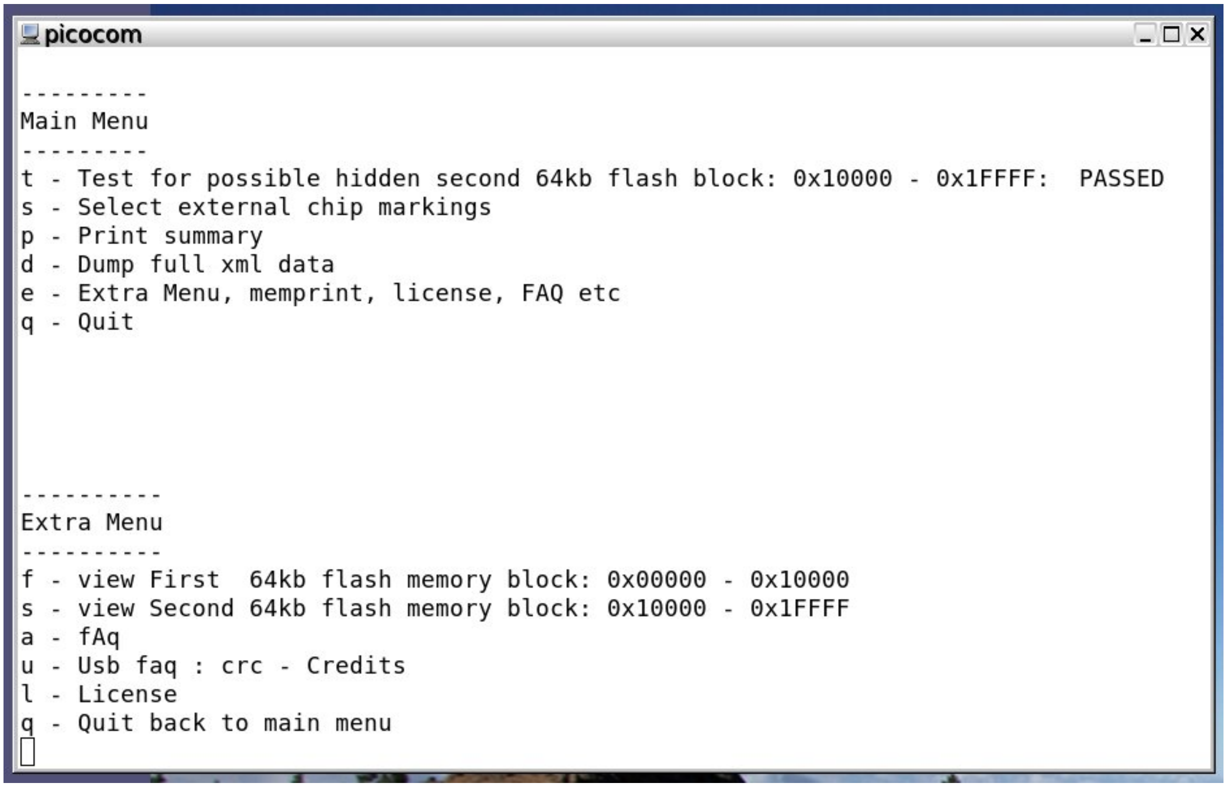
Menues

Main



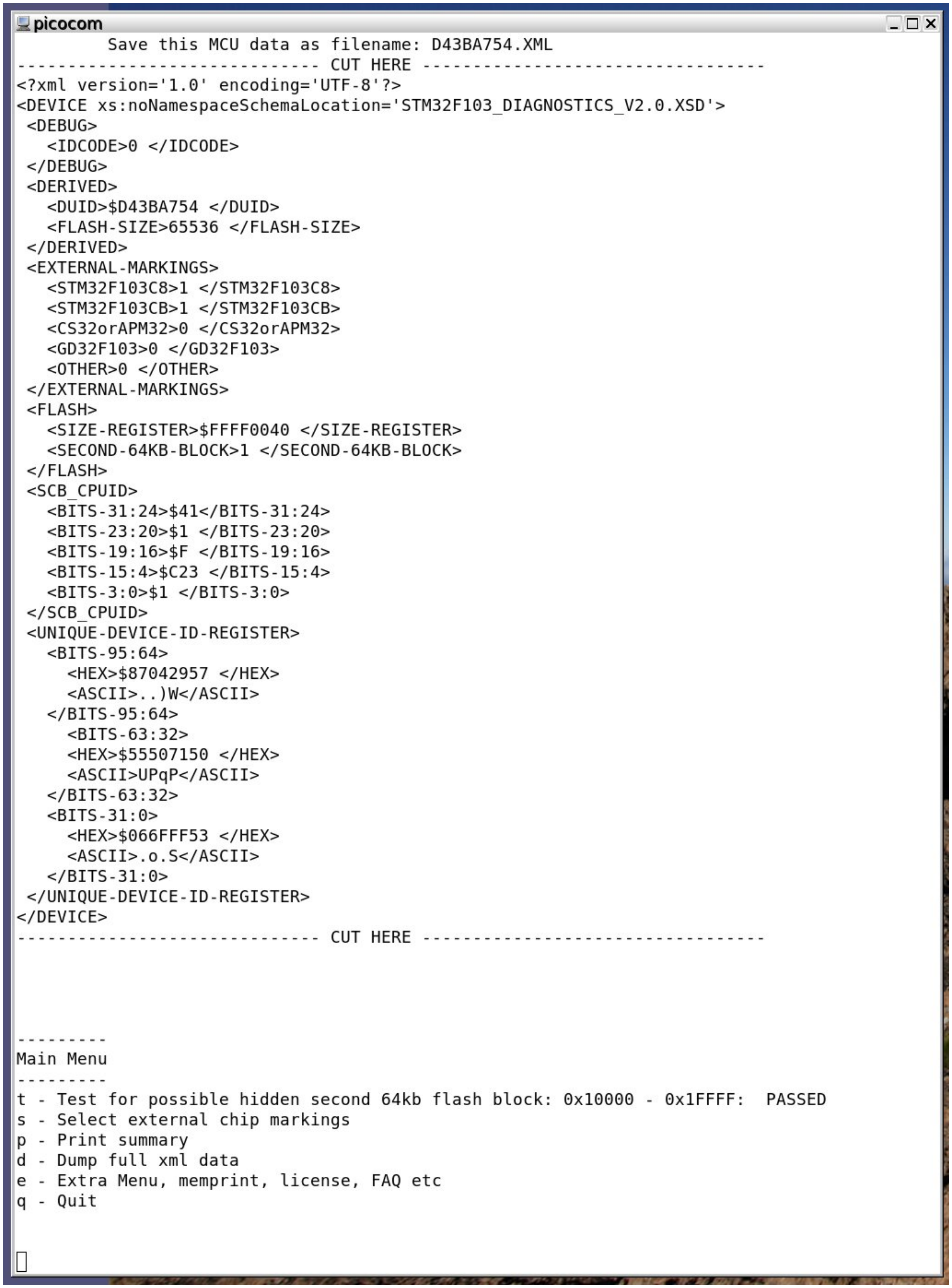
```
-------
Summary
-------
Part: STM32F1 series
Serial Number: D43BA754
External Markings: STM32F103C8
Flash Bytes declared: 65536
128kB Flash present


---------
Main Menu
---------
t - Test for possible hidden second 64kb flash block: 0x10000 - 0x1FFFF:  PASSED
s - Select external chip markings
p - Print summary
d - Dump full xml data
e - Extra Menu, memprint, license, FAQ etc
q - Quit
```

Extra



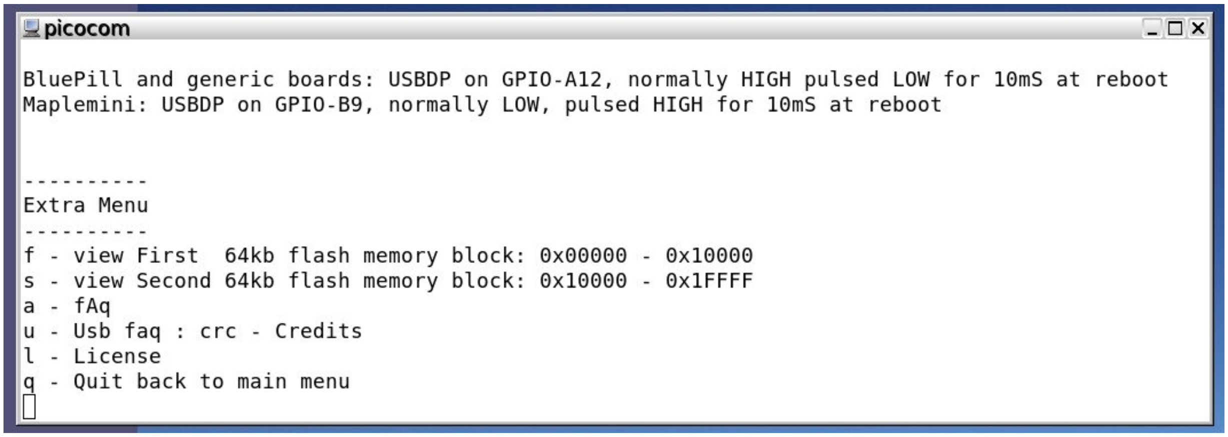
```
---------
Main Menu
---------
t - Test for possible hidden second 64kb flash block: 0x10000 - 0x1FFFF:  PASSED
s - Select external chip markings
p - Print summary
d - Dump full xml data
e - Extra Menu, memprint, license, FAQ etc
q - Quit






----------
Extra Menu
----------
f - view First  64kb flash memory block: 0x00000 - 0x10000
s - view Second 64kb flash memory block: 0x10000 - 0x1FFFF
a - fAq
u - Usb faq : crc - Credits
l - License
q - Quit back to main menu
```

Xml



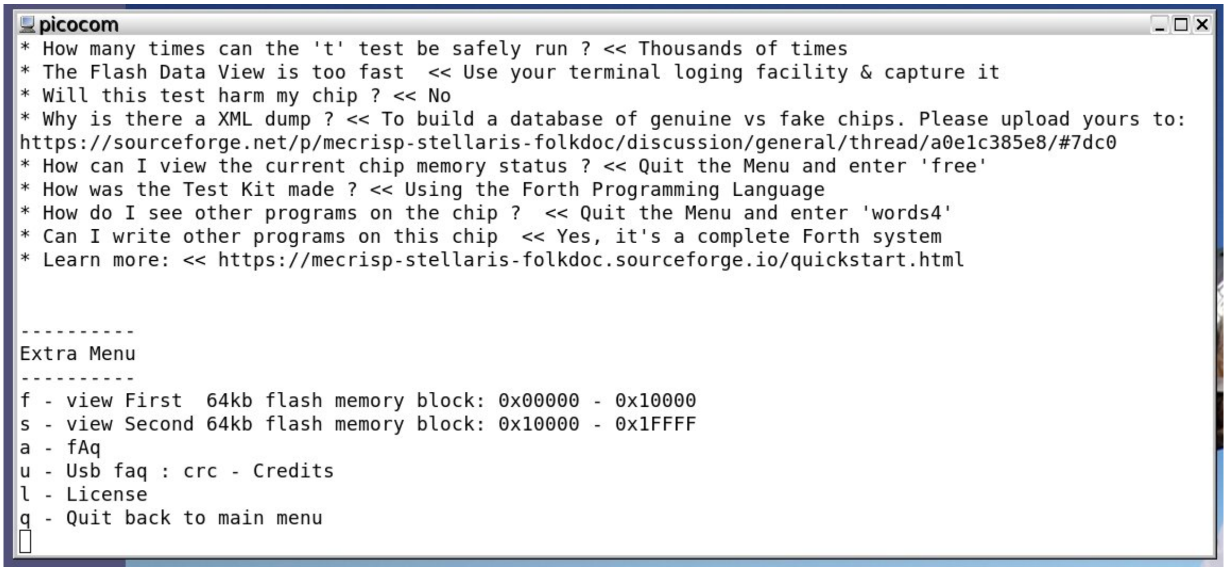
```
         Save this MCU data as filename: D43BA754.XML
------------------------------ CUT HERE ----------------------------------
<?xml version='1.0' encoding='UTF-8'?>
<DEVICE xs:noNamespaceSchemaLocation='STM32F103_DIAGNOSTICS_V2.0.XSD'>
 <DEBUG>
   <IDCODE>0 </IDCODE>
 </DEBUG>
 <DERIVED>
   <DUID>$D43BA754 </DUID>
   <FLASH-SIZE>65536 </FLASH-SIZE>
 </DERIVED>
 <EXTERNAL-MARKINGS>
   <STM32F103C8>1 </STM32F103C8>
   <STM32F103CB>1 </STM32F103CB>
   <CS32orAPM32>0 </CS32orAPM32>
   <GD32F103>0 </GD32F103>
   <OTHER>0 </OTHER>
 </EXTERNAL-MARKINGS>
 <FLASH>
   <SIZE-REGISTER>$FFFF0040 </SIZE-REGISTER>
   <SECOND-64KB-BLOCK>1 </SECOND-64KB-BLOCK>
 </FLASH>
 <SCB_CPUID>
   <BITS-31:24>$41</BITS-31:24>
   <BITS-23:20>$1 </BITS-23:20>
   <BITS-19:16>$F </BITS-19:16>
   <BITS-15:4>$C23 </BITS-15:4>
   <BITS-3:0>$1 </BITS-3:0>
 </SCB_CPUID>
 <UNIQUE-DEVICE-ID-REGISTER>
   <BITS-95:64>
     <HEX>$87042957 </HEX>
     <ASCII>..)W</ASCII>
   </BITS-95:64>
     <BITS-63:32>
     <HEX>$55507150 </HEX>
     <ASCII>UPqP</ASCII>
   </BITS-63:32>
   <BITS-31:0>
     <HEX>$066FFF53 </HEX>
     <ASCII>.o.S</ASCII>
   </BITS-31:0>
 </UNIQUE-DEVICE-ID-REGISTER>
</DEVICE>
------------------------------ CUT HERE ----------------------------------




---------
Main Menu
---------
t - Test for possible hidden second 64kb flash block: 0x10000 - 0x1FFFF:  PASSED
s - Select external chip markings
p - Print summary
d - Dump full xml data
e - Extra Menu, memprint, license, FAQ etc
q - Quit


```

Usb



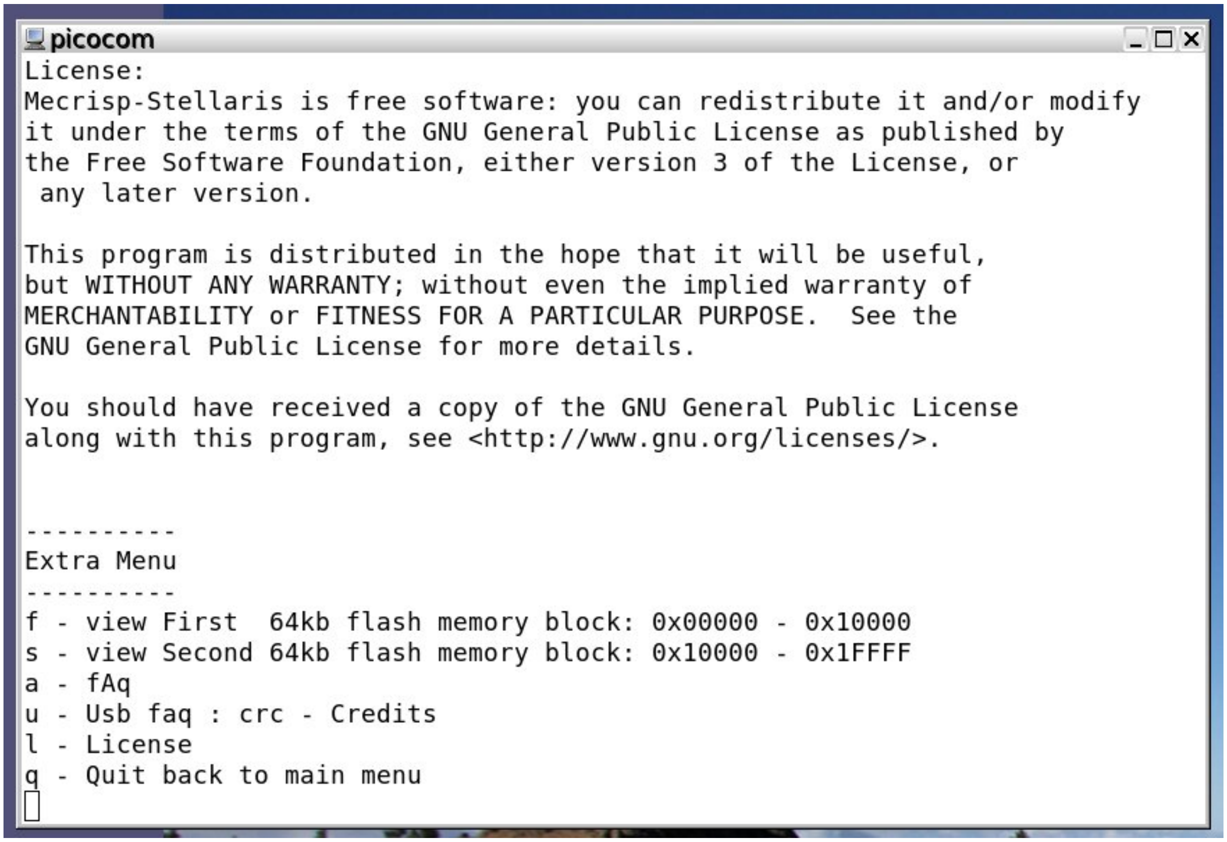
```
BluePill and generic boards: USBDP on GPIO-A12, normally HIGH pulsed LOW for 10mS at reboot
Maplemini: USBDP on GPIO-B9, normally LOW, pulsed HIGH for 10mS at reboot



----------
Extra Menu
----------
f - view First  64kb flash memory block: 0x00000 - 0x10000
s - view Second 64kb flash memory block: 0x10000 - 0x1FFFF
a - fAq
u - Usb faq : crc - Credits
l - License
q - Quit back to main menu
```

Faq



```
* How many times can the 't' test be safely run ? << Thousands of times
* The Flash Data View is too fast  << Use your terminal loging facility & capture it
* Will this test harm my chip ? << No
* Why is there a XML dump ? << To build a database of genuine vs fake chips. Please upload yours to:
https://sourceforge.net/p/mecrisp-stellaris-folkdoc/discussion/general/thread/a0e1c385e8/#7dc0
* How can I view the current chip memory status ? << Quit the Menu and enter 'free'
* How was the Test Kit made ? << Using the Forth Programming Language
* How do I see other programs on the chip ?  << Quit the Menu and enter 'words4'
* Can I write other programs on this chip  << Yes, it's a complete Forth system
* Learn more: << https://mecrisp-stellaris-folkdoc.sourceforge.io/quickstart.html



----------
Extra Menu
----------
f - view First  64kb flash memory block: 0x00000 - 0x10000
s - view Second 64kb flash memory block: 0x10000 - 0x1FFFF
a - fAq
u - Usb faq : crc - Credits
l - License
q - Quit back to main menu
```

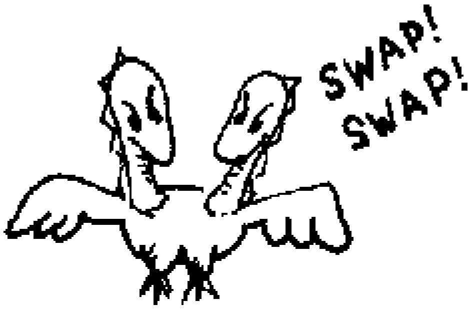
License



```
License:
Mecrisp-Stellaris is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
 any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program, see <http://www.gnu.org/licenses/>.



----------
Extra Menu
----------
f - view First  64kb flash memory block: 0x00000 - 0x10000
s - view Second 64kb flash memory block: 0x10000 - 0x1FFFF
a - fAq
u - Usb faq : crc - Credits
l - License
q - Quit back to main menu
```

Swap Dragon Prize

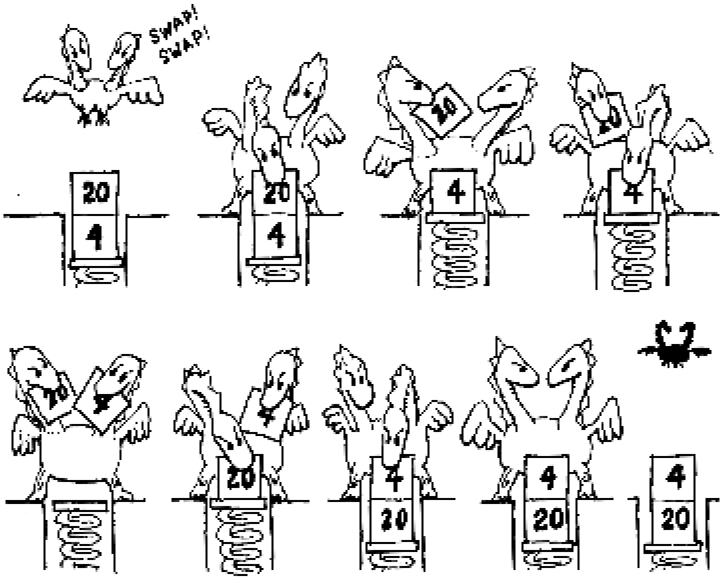
Reference: https://wiki.forth-ev.de/doku.php/en:infos:drachenpreis



Every year the Forth Society e.V. honours an outstanding personality for his/her Forth activities. Started in 1986 as a plastic toy trophy, the SWAP-Dragon has been reborn in 1997 as a beautiful bronze sculpture.

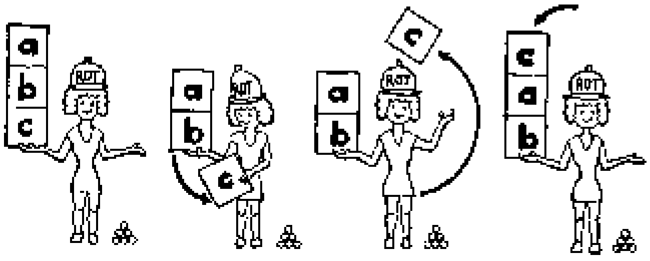
How it all began

When the programming language Forth became publically available in the early 80s, the book Starting Forth by Leo Brodie, formerly Forth Inc., was one of the sources for beginners. It became very popular not only because of the entertaining text but also because the insightful images, that described the different system components and operators as comic figures. To explain the operator SWAP Leo Brodie used a little two-headed dragon, which is today known as the SWAP-Dragon. Below is the original comic from the first edition of Starting Forth that allowed so many of us to understand, what SWAP was supposed to do:


SWAP!
SWAP!
20
4

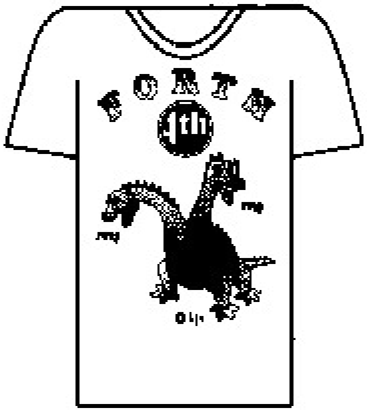
The word SWAP is defined to swich the order of the top two stack items. As with the other stack manipulation operators, you can test SWAP at your terminal in “calculator style”; that is, it doesn’t have to be contained in a definition. - Leo Brodie in Starting Forth 1st ed, p. 44

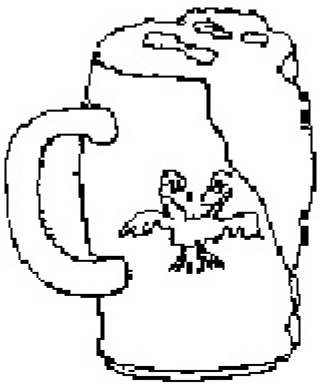
But this is not the only reference to SWAP in Starting Forth. When SWAP sees the operator ROT, which exchanges the upper three entries of the data stack, he shivered over so much complexity:



I HATE JUGGLERS!
ME TOO!

The SWAP becomes the unofficial mascot of the FORTH society. Sometimes he appears on Forth-Shirts and on beer mugs (?) - and always in our heads..





1986 – A prize is born

On its second yearly conference the FORTH society initiated the SWAP-Dragon prize, with which Michael Kalus was honoured for his enormous work in the early days of the Vierten Dimension, the member journal of the FORTH society. When he was presented with a little two-headed toy dragon, he was so impressed that he declared:

When this all lasts for 10 years, I will donate one made from bronze.

Although it is still unclear, what he meant with this all, it did last for more than 10 years and there are still witnesses for this amongst us.

The plastic SWAP dragon had a notable property. It smelled surprisingly of strawberries! Unfortunately we cannot show that here. (So much for multimedia :-) Here a photo of the plastic trophy:

The Dragon council

Then it was realized, that it is a full task to find a person to honour – the Dragon council, consisting out of all former prize recipients was formed. They meet every year during the annual meeting of the FORTH society and deliberate over a new member to bestow the honour on. Last year's prize winner has to contribute at least 3 serious suggestions. The SWAP Dragon bronze sculpture

When it became clear that what was meant with this all, must be older than ten years, the work on a bronze sculpture began. Rolf Kretzschmar, former editor of Vierte Dimension and sculptor, created a wonderful SWAP dragon sculpture, that happily bites into the upper stack entry. Here a photo of the SWAP dragon in bronze:

The winners of the award

Custodian of the SWAP Dragon

Year	Name	Annual meeting location
1986	Michael Kalus	Buchholz/Nordheide
1989	Heinz Schnitter	Aachen
1990	Jörg Staben	Frankfurt
1991	Klaus Schleisiek	München
1992	Ulrike Schnitter	Rostock
1993	Jens Wilke	Nürnberg
1994	Jörg Plewe	Malente-Neukirchen
1995	Friederich Prinz	Berlin

Year	Name	Annual meeting location
1996	Klaus Kohl	Mittweida-Höfchen
1997	Ulrich Hoffmann	Ludwigshafen
1998	Bernd Paysan	Neukirchen-Vluyn
1999	Fred Behringer	Oberammergau
2000	Egmont Woitzel	Hamburg
2001	Martin Bitter	Hamminkeln-Dingden
2002	Hans Eckes	Garmisch-Partenkirchen
2003	Thomas Beierlein	Lambrecht
2004	Ewald Rieger	Burg-Staaken (Fehmarn)
2005	Johannes Reilhofer	Krögis bei Meißen

Year	Name	Annual meeting location
2006	Carsten Strotmann	Witten (Ruhrgebiet)
2007	Anton Ertl	Baden (Wien)
2008	Manfred Mahlow	Kloster Roggenburg (Ulm)
2009	Stephen Pelc	Neuenkirchen (bei Rheine)
2010	Jörg Völker	Rostock-Warnemünde
2011	Friedel Amend	Goslar
2012	Erich Wälde	Beukenhof in Biezenmortel (NL)
2013	Gerald Wodni	Garmisch-Partenkirchen
2014	Willi Stricker	Bad Vöslau
2015	Karsten Roederer	Hannover

Year	Name	Annual meeting location
2016	Thomas Prinz	Augsburg
2017	Klaus Zobawa	Kalkar
2019	Matthias Koch	?

© The Swap drawing and other drawings from Leo Brodie’s Starting Forth are copyrighted by Forth, Inc.

© The Swap Bronze is an artwork done by Rolf Kretzschmar.

© The Forth-Shirt is an artwork done by Heinz Schnitter.

English translation by Sabine Bodzian.

Systick Library

This is a general purpose interrupt driven Systick timing library that should work on any STM32Fxxxx.

It has an accurate blocking delay and event timer.

Note

Systick must be initialised before use and takes the “1ms-cal-value” as a parameter: “<1ms-cal-value> init.systick”

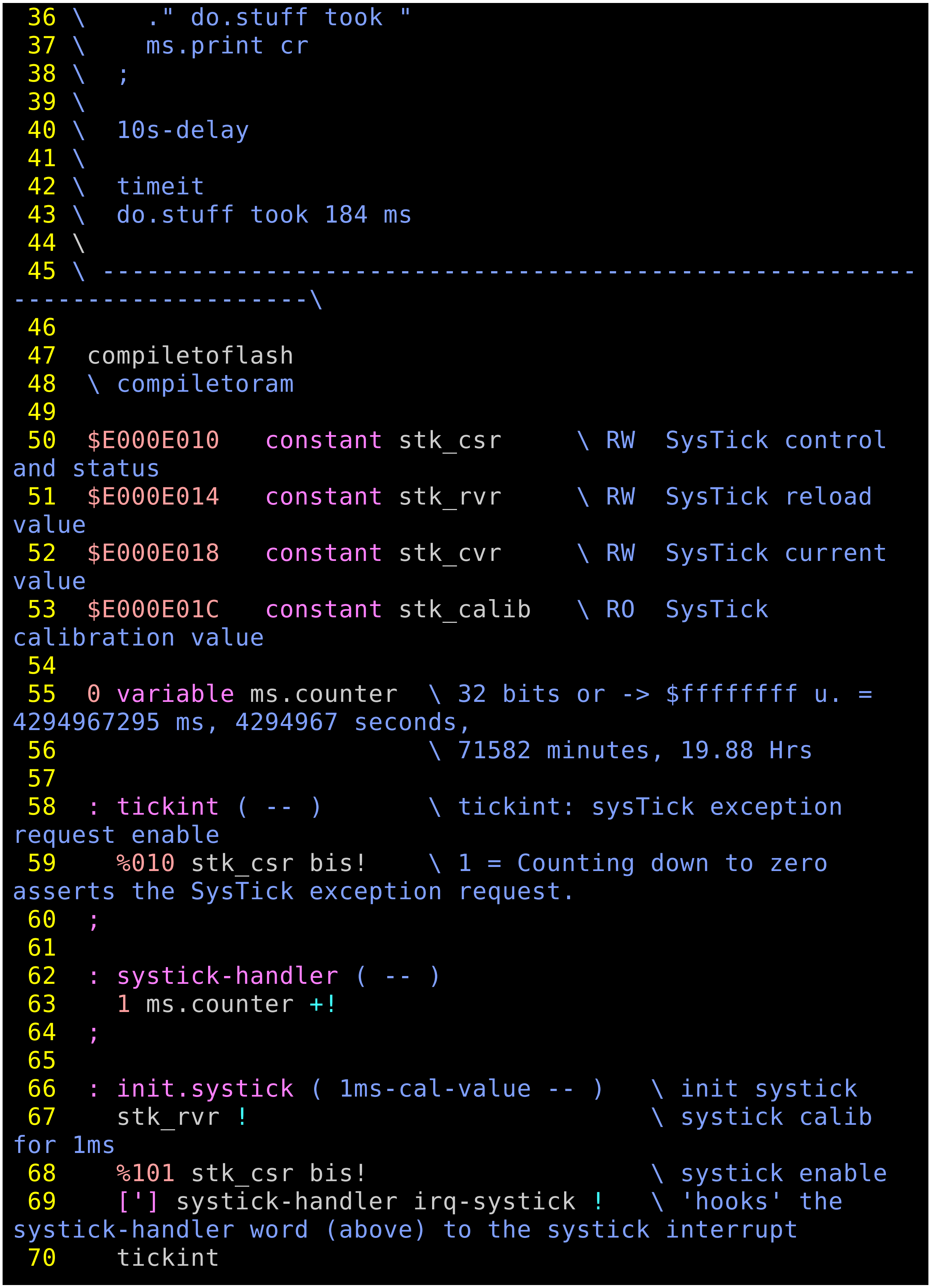
Library Words

Word	Description	Example
ms.delay (u - - delay)	Accurate BLOCKING ms delay	1000 ms.delay
ms.counter.reset (- -)	zero counter	ms.counter.reset
ms.read (- -)	put ms.counter value on Stack	1000 on the stack
ms.print (- -)	print ms.counter to terminal screen	1000 ms
seconds.print (- -)	print seconds to 3 decimal places	Elapsed time is: 1.000 seconds
ms.reached? (u - - yes = -1 | no = 0)	True if count >= u	500 ms.reached?

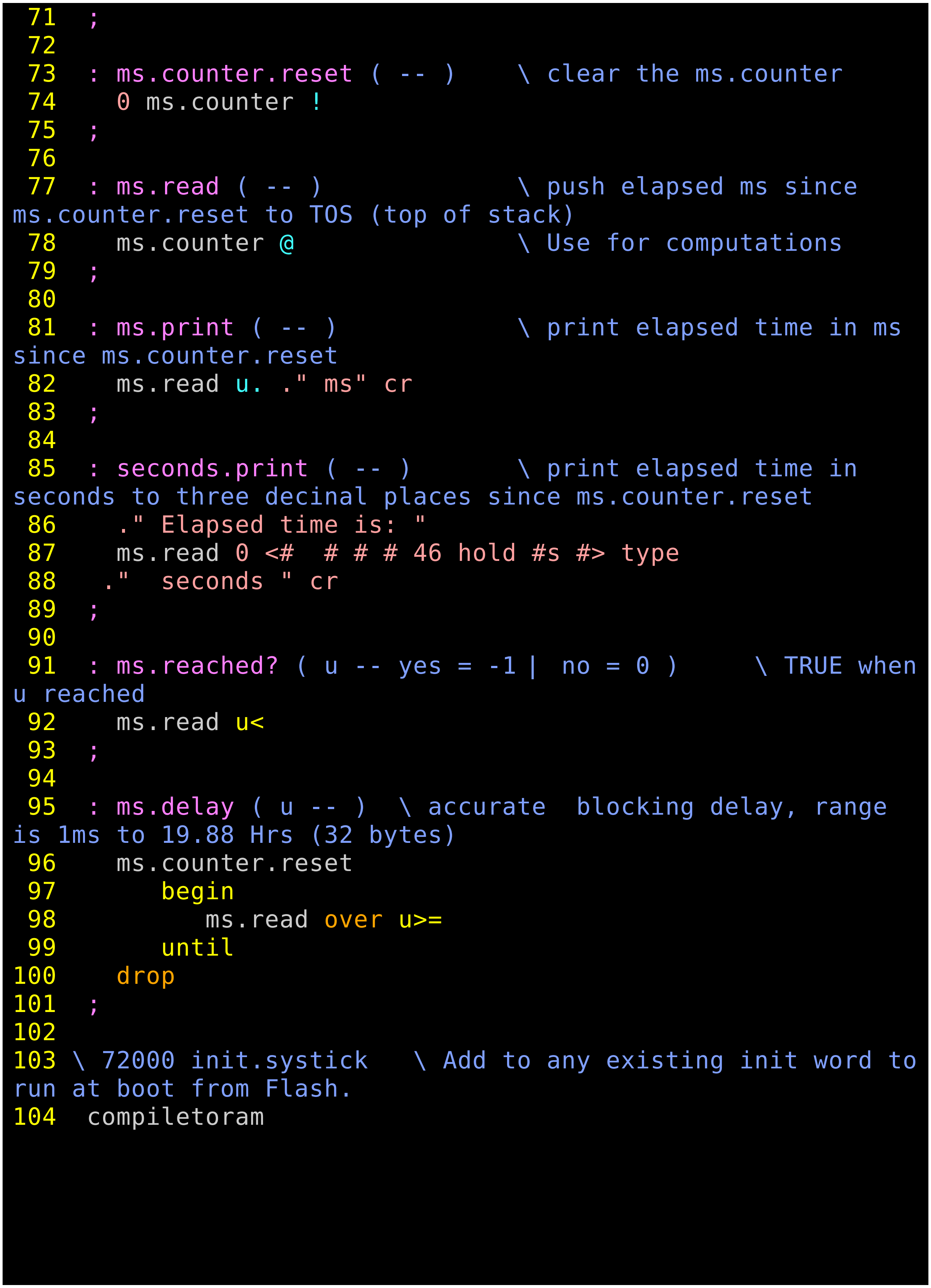
Code

Download systick.fs

```
\ Program Name: systick.fs
\ Date: Wed 8 Jan 2020 13:13:57 AEDT
\ Copyright 2020 by t.j.porter <terry@tjporter.com.au>, licensed under the GPLV3
\ For Mecrisp-Stellaris by Matthias Koch.
\ https://sourceforge.net/projects/mecrisp/
\ Standalone: no preloaded support files required
\
\ This Program: Interrupt driven STM32Fxxx Systick Timing Library
\
\ Note the (STK) is not a part of STM32F CMSIS-SVD
\ See: ST PM0215 Programming manual, page 85-91
\
\ --------------------------- Example 1ms-cal-value -------------------------\
\  8000   Standard Mecrisp-Stellaris using 8MHz internal RC clock
\  48000  STM32F0xx running at 48 MHz
\  72000  STM32F10x running at 72 MHz
\ ------------------------- Example Library Usage ---------------------------\
\
\  Systick must be initialised before use: "<1ms-cal-value> init.systick"
\
\ ---------------------------------------------------------------------------\
\
\  : 10s-delay cr          \ Demo 10 second delay
\    ." 10 second delay starting, please wait ..." cr
\    10000 ms.delay   ( 10 seconds )
\    ." 10 second delay finished" cr
\  ;
\
\  : do.stuff ( -- )       \ blocking do-loop
\    4800000 0 do loop     \ time will vary depending on interrupts etc so
\  ;                       \ blocking do-loop are not very accurate
\
\  : timeit cr             \ measure the time taken to "do.stuff" above
\    ms.counter.reset
\    do.stuff
```



```
\    ." do.stuff took "
\    ms.print cr
\  ;
\
\  10s-delay
\
\  timeit
\  do.stuff took 184 ms
\
\ ---------------------------------------------------------------------------\

 compiletoflash
 \ compiletoram

 $E000E010   constant stk_csr     \ RW  SysTick control and status
 $E000E014   constant stk_rvr     \ RW  SysTick reload value
 $E000E018   constant stk_cvr     \ RW  SysTick current value
 $E000E01C   constant stk_calib   \ RO  SysTick calibration value

 0 variable ms.counter  \ 32 bits or -> $ffffffff u. = 4294967295 ms, 4294967 seconds,
                        \ 71582 minutes, 19.88 Hrs

 : tickint ( -- )       \ tickint: sysTick exception request enable
   %010 stk_csr bis!    \ 1 = Counting down to zero asserts the SysTick exception request.
 ;

 : systick-handler ( -- )
   1 ms.counter +!
 ;

 : init.systick ( 1ms-cal-value -- )   \ init systick
   stk_rvr !                           \ systick calib for 1ms
   %101 stk_csr bis!                   \ systick enable
   ['] systick-handler irq-systick !   \ 'hooks' the systick-handler word (above) to the systick interrupt
   tickint
```



```
 ;

 : ms.counter.reset ( -- )    \ clear the ms.counter
   0 ms.counter !
 ;

 : ms.read ( -- )             \ push elapsed ms since ms.counter.reset to TOS (top of stack)
   ms.counter @               \ Use for computations
 ;

 : ms.print ( -- )            \ print elapsed time in ms since ms.counter.reset
   ms.read u. ." ms" cr
 ;

 : seconds.print ( -- )       \ print elapsed time in seconds to three decinal places since ms.counter.reset
   ." Elapsed time is: "
   ms.read 0 <#  # # # 46 hold #s #> type
  ."  seconds " cr
 ;

 : ms.reached? ( u -- yes = -1 |  no = 0 )     \ TRUE when u reached
   ms.read u<
 ;

 : ms.delay ( u -- )  \ accurate  blocking delay, range is 1ms to 19.88 Hrs (32 bytes)
   ms.counter.reset
      begin
         ms.read over u>=
      until
   drop
 ;

\ 72000 init.systick   \ Add to any existing init word to run at boot from Flash.
 compiletoram
```

Temperature Sensor Simple

This is a simpler version of my first LMT01 temperature sensor project which used:

1. Two Interrupts
2. A Timer
3. Supported multiple LMT01 sensors

This version uses a Blocking Delay instead of a Timer, one interrupt and it only supports a single LMT01 sensor.

This is also my first project using two files where:

1. One file is for configuration only. Its filename ends in *.includes.fs and must be loaded first. This file provides all prerequisites for the second file below
2. The second file contains Words pertaining to the higher level functions of the program itself.

I’m trying this strategy to separate the two functions of any embedded program, namely “configuration” and “program” in order to make the project more readable.

Downloads

These are all the project files including a ready to flash binary for a STM32F0 Discovery Board.

1. **`f0-lmt01-temp-sensor-simple-version.fs`**
2. **`f0-lmt01-temp-sensor-simple-version.includes.fs`**
3. **`Standalone binary file containing everything needed for this project which will run on powerup`**
4. **`Binary file README`**

5. **`List of Words in the Binary above`**

Warning

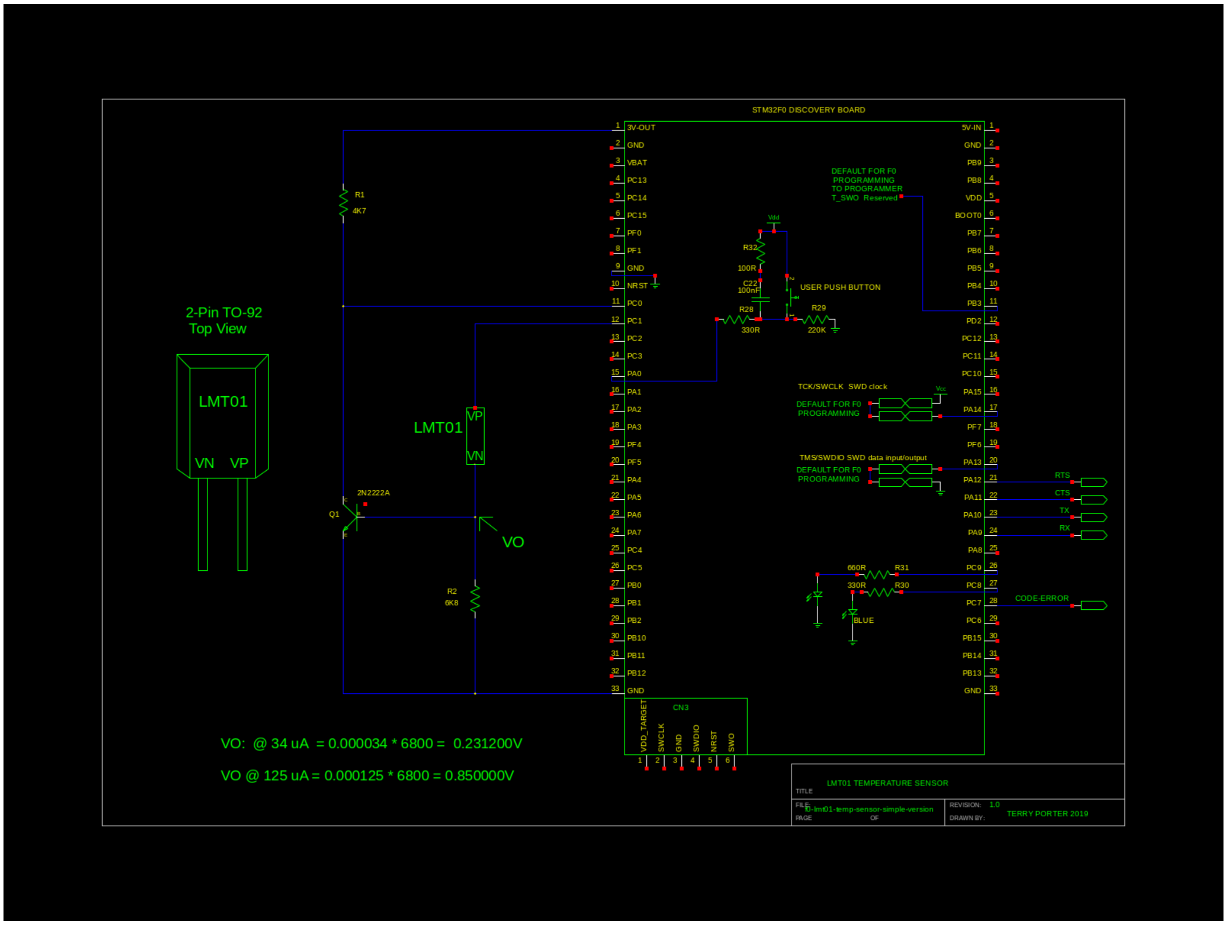
The Standalone Binary has RTS/CTS handshaking. Connect to a 3.3v/usb dongle which supports them for correct operation. Terminal should be set to 460800 baud.Connect PA11 (CTS) to 0V (ground), ignore PC7 (code-error). PA12 (RTS) connects to the dongle CTS. PA10 (RX) connects to dongle TX, etc

Operation

Flash the board with the standalone binary file above, connect the FOUR parts to the Discovery Board as shown in the schematic below and type “t” in the terminal to read the temperature in Degrees C.

```
t
23.2C   ok.
```

Schematic


STM32F0 DISCOVERY BOARD
2-Pin TO-92
Top View
LMT01
VN
VP
LMT01
VP
VN
R1
4K7
2N2222A
Q1
VO
R2
6K8
3V-OUT
GND
VBAT
PC13
PC14
PC15
PF0
PF1
GND
NRST
PC0
PC1
PC2
PC3
PA0
PA1
PA2
PA3
PF4
PF5
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PB10
PB11
PB12
GND
5V-IN
GND
PB9
PB8
VDD
BOOT0
PB7
PB6
PB5
PB4
PB3
PD2
PC12
PC11
PC10
PA15
PA14
PF7
PF6
PA13
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
GND
DEFAULT FOR F0
PROGRAMMING
TO PROGRAMMER
T_SWO Reserved
Vdd
R32
100R
C22
100nF
USER PUSH BUTTON
R28
330R
R29
220K
TCK/SWCLK SWD clock
Vcc
DEFAULT FOR F0
PROGRAMMING
TMS/SWDIO SWD data input/output
DEFAULT FOR F0
PROGRAMMING
RTS
CTS
TX
RX
660R
R31
330R
R30
BLUE
CODE-ERROR
CN3
VDD_TARGET
SWCLK
GND
SWDIO
NRST
SWO
VO: @ 34 uA = 0.000034 * 6800 = 0.231200V
VO @ 125 uA = 0.000125 * 6800 = 0.850000V
LMT01 TEMPERATURE SENSOR
TITLE
FILE: f0-lmt01-temp-sensor-simple-version
PAGE
OF
REVISION: 1.0
DRAWN BY:
TERRY PORTER 2019

Note

All parts (including wiring) in the big green rectangle are INSIDE the Discovery Board, you do not add them. This project has only 4 parts to be connected and they are on the left of the rectangle.

How Does It Work ?

1. PC1 is raised high which provides power to the LMT01 temperature sensor.
2. The LMT01 sensor
    a. Powers up and determines the temperature.
    b. Sends the temperature as a series of current pulses representing digital “high” and “low”.

   c. Q1 converts the current pulses into digital “1” and “0” voltage pulses.

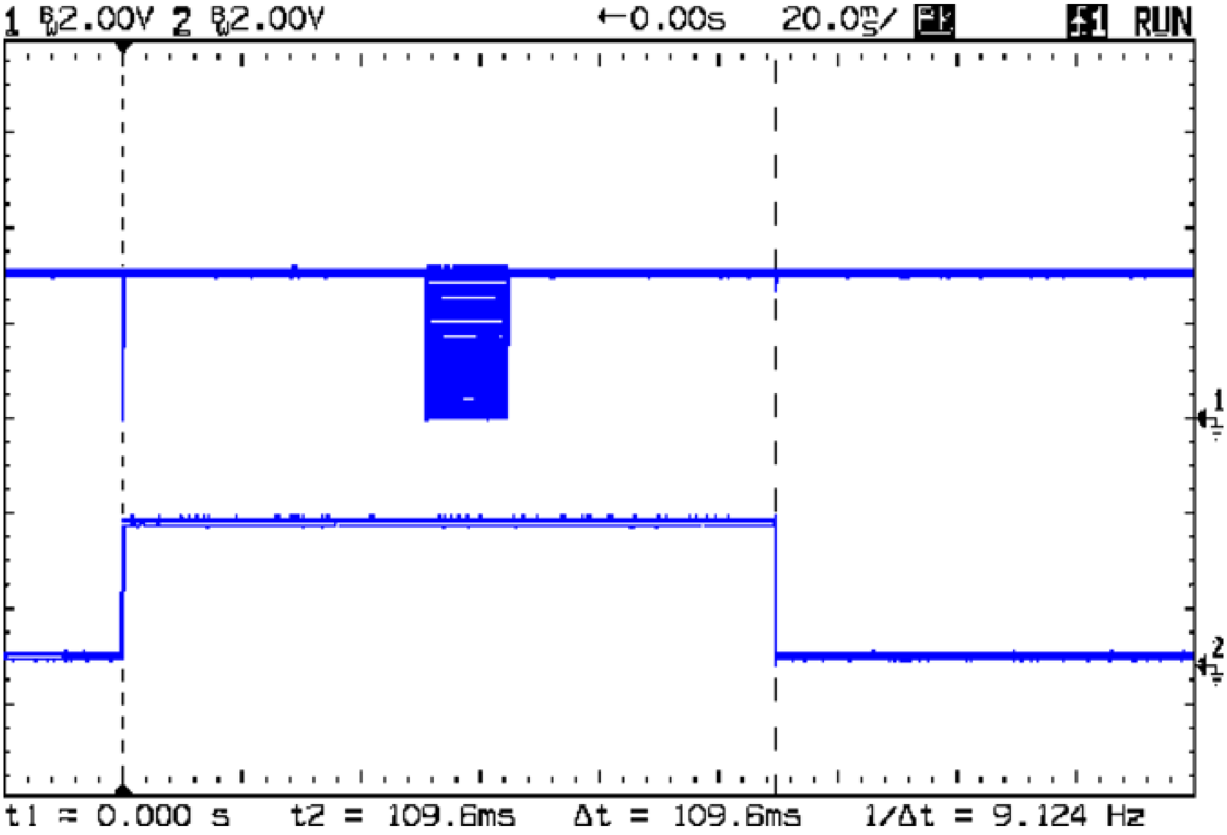
4. PCO, configured as a INPUT triggers a interrupt on each falling edge of the received pulses. These pulses are counted by the interrupt handler.
5. After PC1 is raised high, a blocking delay of 105 milliseconds is initiated.
6. The 105 millisecond delay allows enough time for all the pulses to be counted.
7. When the 105 millisecond delay is finished, power to the LMT01 is removed and the temperature is calculated and displayed.

Waveforms

The Scope picture below shows:

1. Waveform on channel 1 is the block of pulses from the LMT01 as received at PC0.
2. Waveform on channel 2 is the power to the LMT01 from PC1.

Waveform Image



Scope settings

```
        State   Volts/Div   Position    Cplg  BW Lim Inv
Probe
Chan 1  On      2.000 V     0.000 V     DC    On     Off
10:1
Chan 2  On      2.000 V     -5.188 V    DC    On     Off
10:1

                      Main        Main        Time  Delayed
Delayed
            Mode      Time/Div    Delay       Ref   Time/Div
Delay
Horizontal  Normal    20.00ms/    0.000 s     Left  --------
-----

Trigger Mode      Source  Level     Holdoff  Slope    Couplg
Reject  NoiseRej
        Normal    Ch 1    2.375 V   16.40ms  Pos      DC
Off     Off
```

```
Display Mode: Peak Det

Cursors: t1=0.000 s     t2=109.6ms     V1(1)=3.062 V
V2(1)=3.000 V
```

TODO LIST

SVD2FORTH

- 11May2020: Make new Stm32 release with latest mods and new html readme

STM32F103

- 07may2020: create new bitfield type combining both CNF and MODE for easy GPIO config.

STM32F051

- 07may2020: Release latest version with color enabled, USART1 @406800 Baud with RTS, colorstack

Rants

So you think you’re a ‘hobbyist’ ?

Why I hate AVR

Why do programmers feel the need to turn statement 1 into statement 2 ?

statement 1

```
GPIOC_BSRR_BR15
```

stetement 2

```
15 port-c gpio-unset-pin
```

The 5 stages of embedded C programmer grief

STM32 TOUCH SENSOR (TSC) DEMO BINARY

This project supplies a binary file ready to run on a STM32F0 Discovery Board. It provides a fully configured Touch Sensor and Menu so that various user created touch sensors may be tested and calibrated.

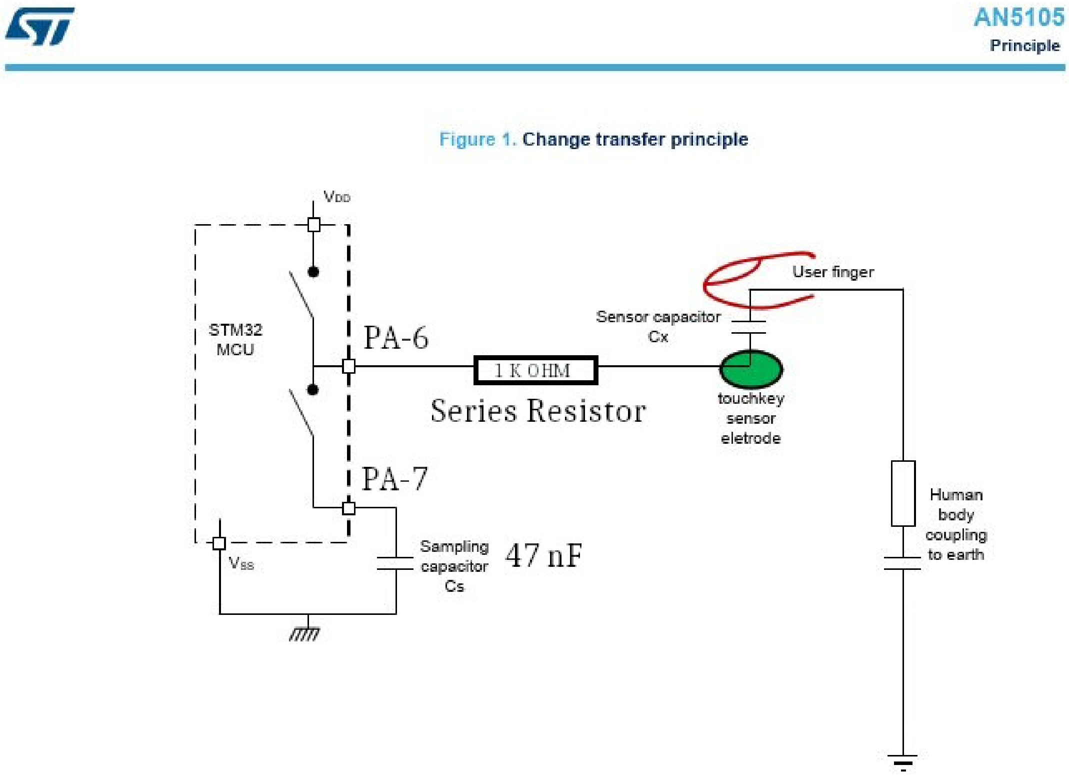
Download: https://sourceforge.net/projects/mecrisp-stellaris-folkdoc/files/STM32F0-Discovery-TouchSensor-Demo.bin

STM Technical Manual Description

The touch sensing controller provides a simple solution for adding capacitive sensing functionality to any application. Capacitive sensing technology is able to detect finger presence near an electrode which is protected from direct touch by a dielectric (for example glass, plastic). The capacitive variation introduced by the finger (or any conductive object) is measured using a proven implementation based on a surface charge transfer acquisition principle.

Schematic

Figure 1. modified with the component values and GPIO Pins I have used in this project.



AN5105
Principle

Figure 1. Change transfer principle

VDD
STM32
MCU
PA-6
PA-7
1 K OHM
Series Resistor
User finger
Sensor capacitor
Cx
touchkey
sensor
eletrode
Human
body
coupling
to earth
Sampling
capacitor
Cs
47 nF
VSS

Documentation

- *STM32F051 Technical Manual*: RM0091
- *Getting started with touch sensing control on STM32 microcontrollers*: AN5105
- *Mecrisp-Stellaris Forth*: This site!

Instructions

- Download the binary and flash it to your F0 Discovery board.
- Connect a serial terminal as shown in the wiring chart below
- Connect the 47 nF capacitor and a touch pad of some kind. A coin with a wire attached and wrapped in stickytape should work fine.
- When the board is powered up you should see the ‘Main Menu’. If not, enter “?” to start the menu.
- Select ‘s’ and scanning should begin ready for touch experiments.

Connection Information

STM32F0 Discovery Board

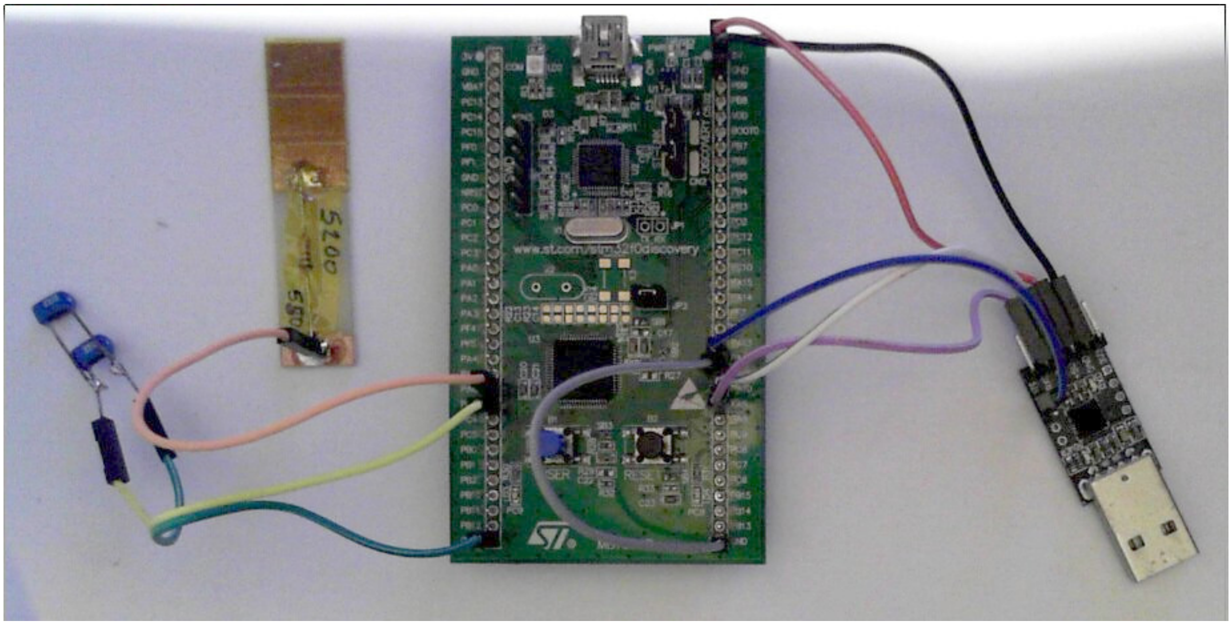
Information

USB-3.3 Volt Dongle

Information

RST
RI
3V3
SUS
SUS
GND
+5V
TXD
RXD
DTR
CTS
RTS
DSR
DCD

Wiring


www.st.com/stm32f0discovery
5200
580
USER
RESET

Wiring Chart

STM32F0 Discovery	Description	USB-3.3v dongle	47 nF Capacitor	Electrode

STM32F0 Discovery	**Description**	**USB-3.3v dongle**	**47nF Capacitor**	**Electrode**
+5V (input)	+5 volts	+5V (output)		
GND	0 volts	GND		
PA9 (TX)	serial data	RX		
PA10 (RX)	serial data	TX		
PA11 (CTS)	handshaking	RTS (not used)		
PA12 (RTS)	handshaking	CTS		
PA7	47 nf Cap		•	
GND	47 nf Cap		•	
PA6	Electrode			•

Notes

- The ELECTRODE has only ONE wire
- The 47 nanoFarad capacitor is non-polarised

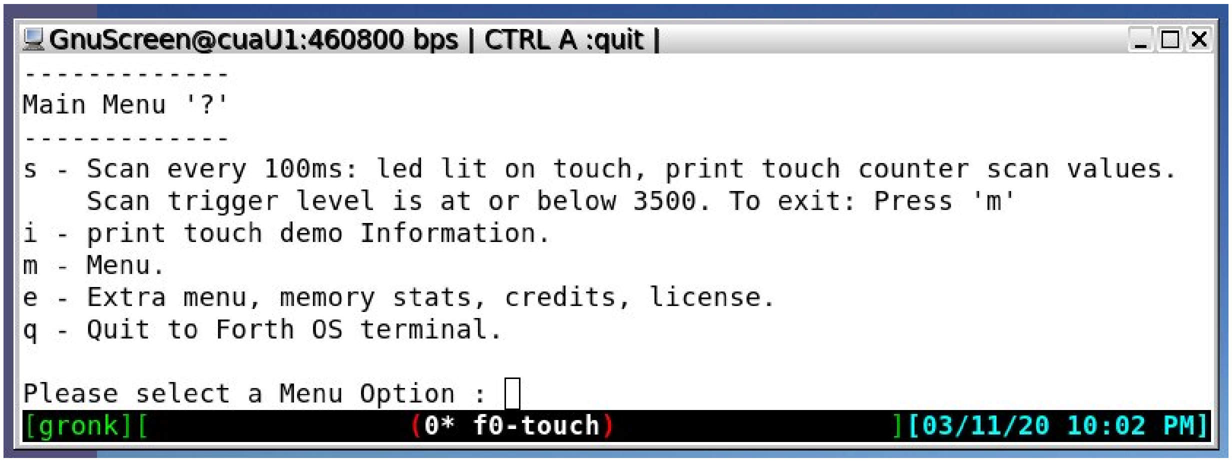
Serial

- 460800 baud, N,1
- Handshaking isn’t essential if you only want to read data from the board (test different electrodes etc). It’s only ESSENTIAL if you want to upload Forth Source.

- CTS/RTS handshaking works but isn't needed so PA11 can connect to GND instead of the USB-3.3v dongle if you like. DON'T leave PA11 FLOATING or serial comms won't work!

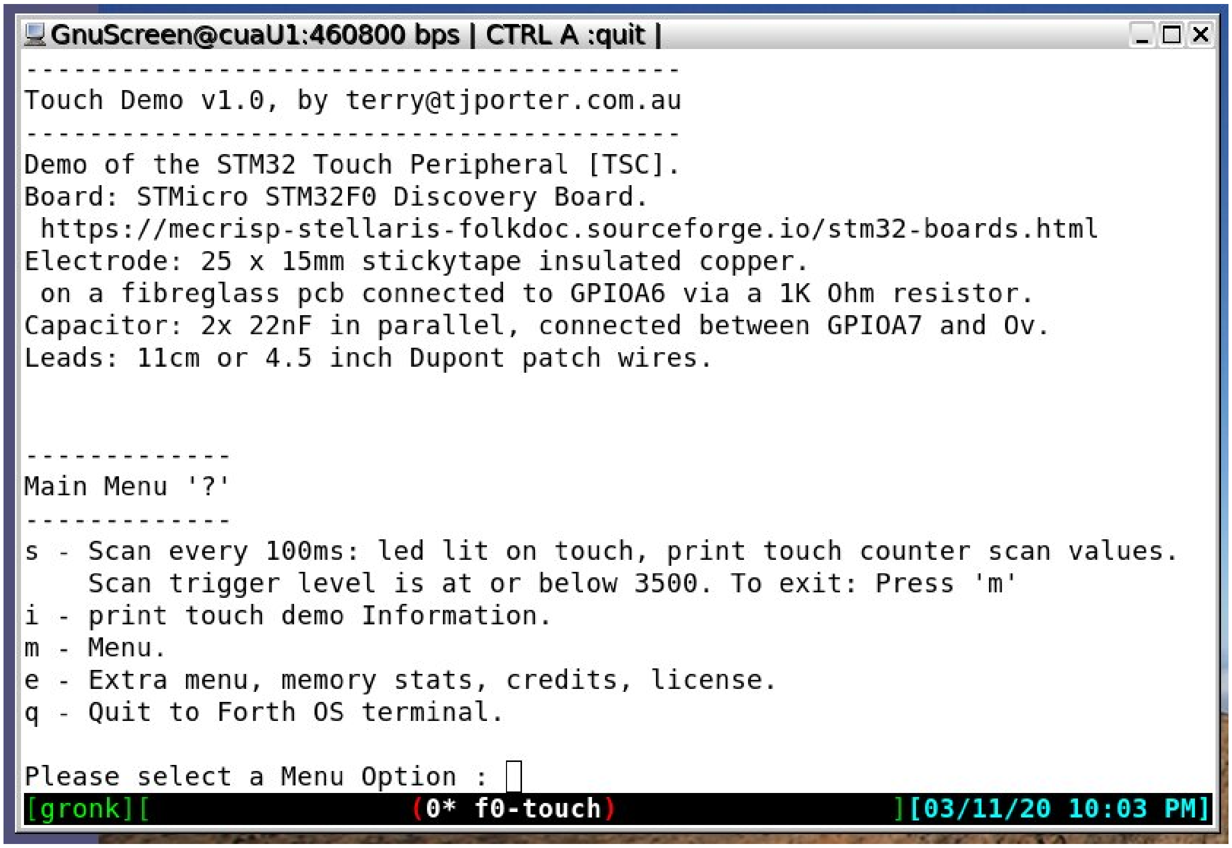
Development Binary Menu

Main Menu



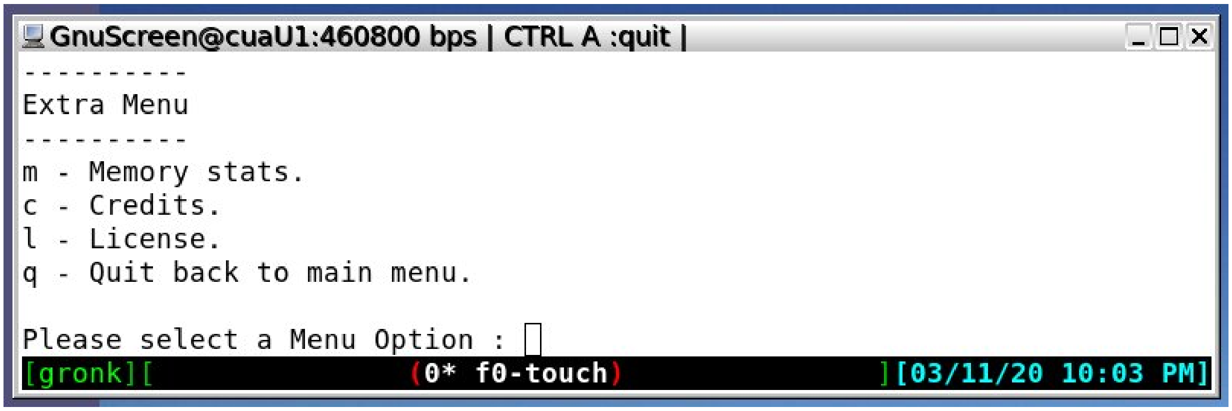
```
-------------
Main Menu '?'
-------------
s - Scan every 100ms: led lit on touch, print touch counter scan values.
    Scan trigger level is at or below 3500. To exit: Press 'm'
i - print touch demo Information.
m - Menu.
e - Extra menu, memory stats, credits, license.
q - Quit to Forth OS terminal.

Please select a Menu Option : 
[gronk][                    (0* f0-touch)                    ][03/11/20 10:02 PM]
```

Information Menu



```
---------------------------------------
Touch Demo v1.0, by terry@tjporter.com.au
---------------------------------------
Demo of the STM32 Touch Peripheral [TSC].
Board: STMicro STM32F0 Discovery Board.
 https://mecrisp-stellaris-folkdoc.sourceforge.io/stm32-boards.html
Electrode: 25 x 15mm stickytape insulated copper.
 on a fibreglass pcb connected to GPIOA6 via a 1K Ohm resistor.
Capacitor: 2x 22nF in parallel, connected between GPIOA7 and Ov.
Leads: 11cm or 4.5 inch Dupont patch wires.



-------------
Main Menu '?'
-------------
s - Scan every 100ms: led lit on touch, print touch counter scan values.
    Scan trigger level is at or below 3500. To exit: Press 'm'
i - print touch demo Information.
m - Menu.
e - Extra menu, memory stats, credits, license.
q - Quit to Forth OS terminal.

Please select a Menu Option :
[gronk][                (0* f0-touch)                ][03/11/20 10:03 PM]
```

Extra Menu



```
----------
Extra Menu
----------
m - Memory stats.
c - Credits.
l - License.
q - Quit back to main menu.

Please select a Menu Option :
[gronk][                (0* f0-touch)                ][03/11/20 10:03 PM]
```

The menu contains most of the functions one needs to test various capacitive sensor designs including the ‘counter’ values which change with sensor size,shape and touch.

A LED on the Discovery Board also lights when a finger ‘press’ is detected.

Menu “s” example

The change in ‘count value’ can be seen in the example below as my finger is placed on and off the touchpad. Readings followed by a “*” mean this was detected as a “touch”. The ratio of not-touched to touched can be as high as 10.8:1.

```
4921
5067
5068
975 *
863 *
1149 *
1149 *
5038
5065
942 *
942 *
5068
5071
5075
5073
5072
5065
5069
5050
998 *
915 *
892 *
884 *
898 *
898 *
4235
5026
```

Touchkey Sensor Electrodes

These are some of my prototypes made from single sided copper coated scrap pcb material.

Writing on Touch Pads?

- Large number: untouched
- Small number: finger pressing on the copper pad.

All pads are insulated by tape (even if it isn’t visible), the finger can’t electrically touch anything.

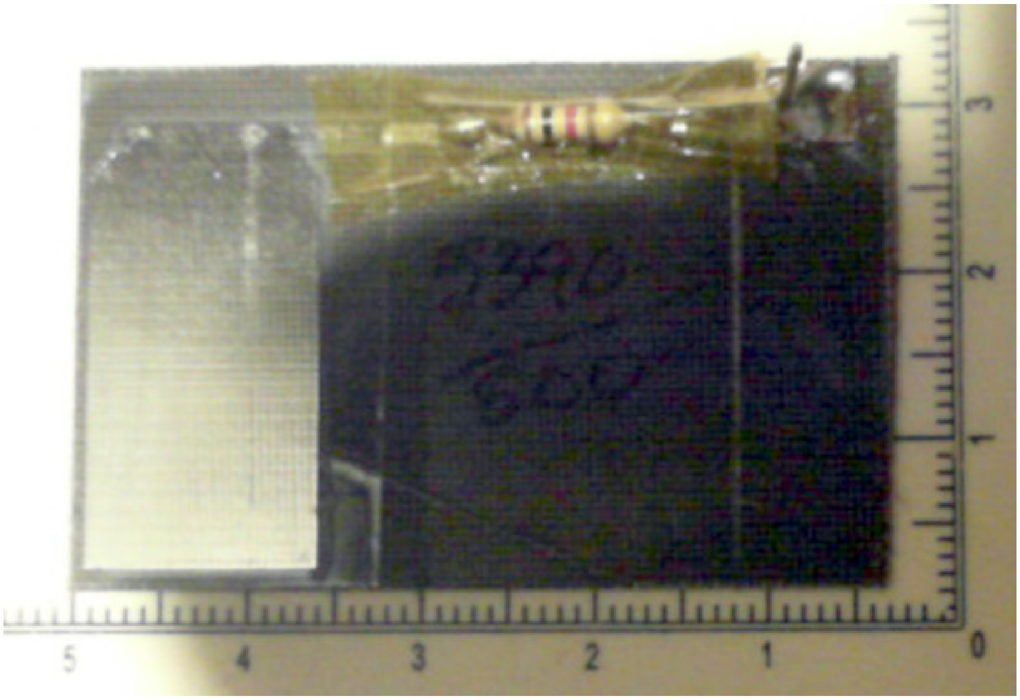
Note

The use of Kapton resulted in slightly less signal/noise ratio.

Scale is in millimeters

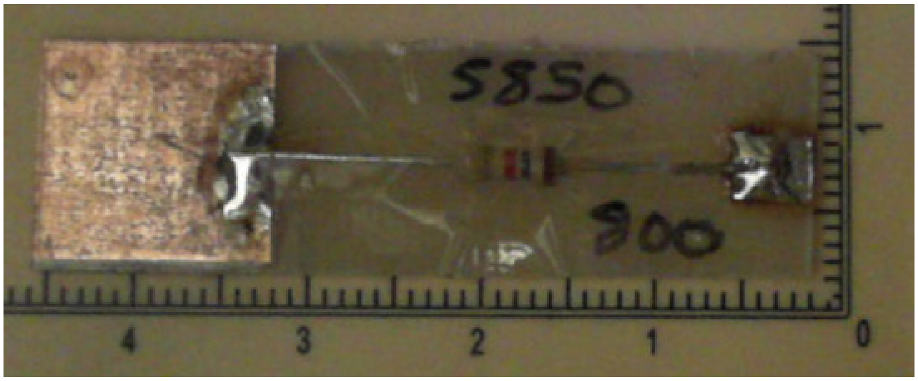
PCB Number	PCB Material	Insulation	Untouched	Touched	Signal/Noise Ratio
1	Teflon Composite	stickytape	5390	500	10.8
2	Fibreglass	stickytape	5850	800	10.8
3	Fibreglass	Kapton	5200	550	9.45

#1 Teflon Composite



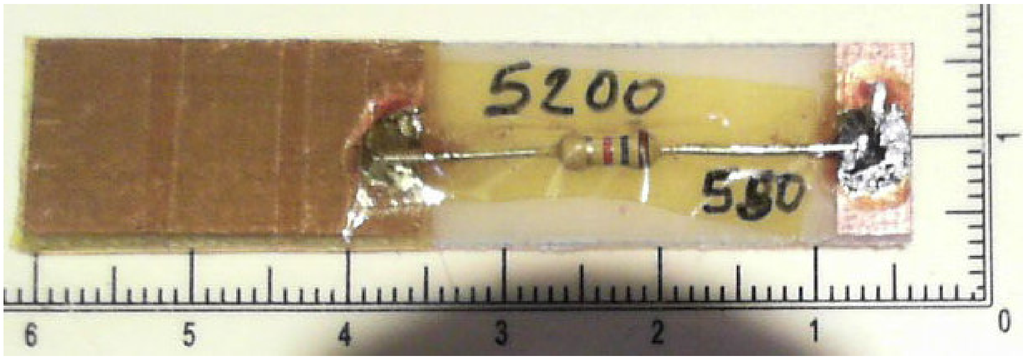
#2 Fibreglass

Insulation: stickytape


5850
800
4
3
2
1
0

#3 Fibreglass

Insulation: Kapton


5200
580
6
5
4
3
2
1
0

Problems ?

No Electrode connected

“s” output:

```
9235
9235
9240
9235
9238
9239
9234
9236
9236
```

No Capacitor connected

“s” output:

```
1 *
1 *
1 *
1 *
1 *
1 *
1 *
```

No Capacitor or Electrode connected

“s” output:

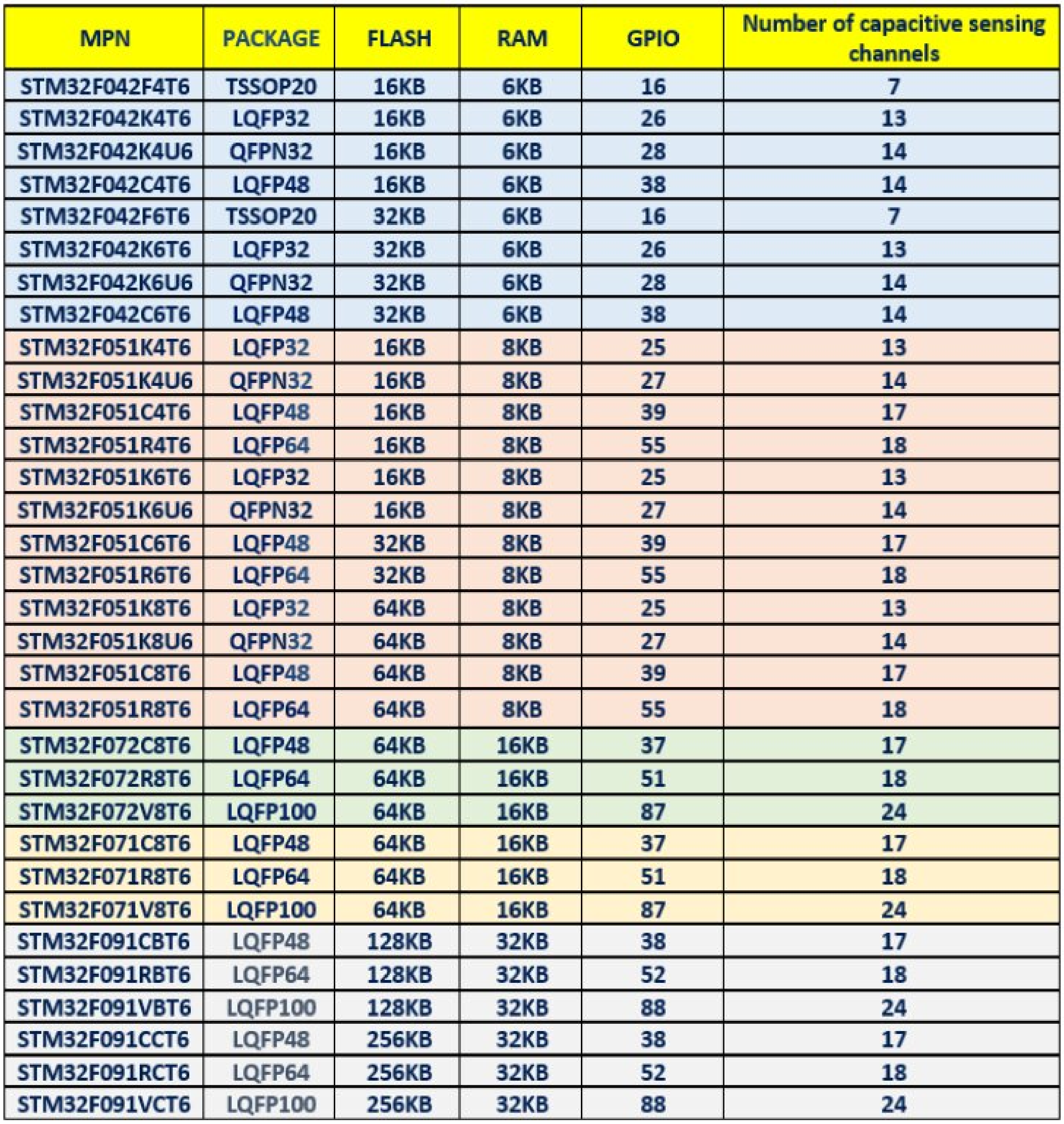
```
1 *
1 *
1 *
1 *
1 *
1 *
1 *
```

The TSC Peripheral

This peripheral is available on many STM32F/L0, and STM32F/L4’s but not all STM’s including the STM32F1xx, eg the ‘Blue Pill’.

touch keys.

Below a list of **STM32F0** that show the number of touch that manage.



MPN	PACKAGE	FLASH	RAM	GPIO	Number of capacitive sensing channels
STM32F042F4T6	TSSOP20	16KB	6KB	16	7
STM32F042K4T6	LQFP32	16KB	6KB	26	13
STM32F042K4U6	QFPN32	16KB	6KB	28	14
STM32F042C4T6	LQFP48	16KB	6KB	38	14
STM32F042F6T6	TSSOP20	32KB	6KB	16	7
STM32F042K6T6	LQFP32	32KB	6KB	26	13
STM32F042K6U6	QFPN32	32KB	6KB	28	14
STM32F042C6T6	LQFP48	32KB	6KB	38	14
STM32F051K4T6	LQFP32	16KB	8KB	25	13
STM32F051K4U6	QFPN32	16KB	8KB	27	14
STM32F051C4T6	LQFP48	16KB	8KB	39	17
STM32F051R4T6	LQFP64	16KB	8KB	55	18
STM32F051K6T6	LQFP32	16KB	8KB	25	13
STM32F051K6U6	QFPN32	16KB	8KB	27	14
STM32F051C6T6	LQFP48	32KB	8KB	39	17
STM32F051R6T6	LQFP64	32KB	8KB	55	18
STM32F051K8T6	LQFP32	64KB	8KB	25	13
STM32F051K8U6	QFPN32	64KB	8KB	27	14
STM32F051C8T6	LQFP48	64KB	8KB	39	17
STM32F051R8T6	LQFP64	64KB	8KB	55	18
STM32F072C8T6	LQFP48	64KB	16KB	37	17
STM32F072R8T6	LQFP64	64KB	16KB	51	18
STM32F072V8T6	LQFP100	64KB	16KB	87	24
STM32F071C8T6	LQFP48	64KB	16KB	37	17
STM32F071R8T6	LQFP64	64KB	16KB	51	18
STM32F071V8T6	LQFP100	64KB	16KB	87	24
STM32F091CBT6	LQFP48	128KB	32KB	38	17
STM32F091RBT6	LQFP64	128KB	32KB	52	18
STM32F091VBT6	LQFP100	128KB	32KB	88	24
STM32F091CCT6	LQFP48	256KB	32KB	38	17
STM32F091RCT6	LQFP64	256KB	32KB	52	18
STM32F091VCT6	LQFP100	256KB	32KB	88	24

Summary

Initially I thought that the TSC results might be a bit flaky or not work at all on a standard low cost “F0 Discovery” board where the lead capacitances are ‘whatever’ and there is a lot of electrical noise.

I couldn’t have been more wrong! Instead I found that the TSC is highly robust and repeatable. I intend to use it in further projects, some of which I only saw the possibilities after playing with this kit.

For Developers

Binary, Brief Description

This project uses Mecrisp-Stellaris Forth on a FO Discovery Board containing a STM32F051 MCU overclocked to 75 MHz (via the Disco 8MHz xtal) and running a simple task scheduler consisting of Task 1 which occurs every 100 milliseconds and triggers a TSC ‘acquisition’ interrupt. Five milliseconds after the acquisition, Task 2 runs displaying the results.

The controlling serial terminal response is the same whether scanning or not.

After quitting out of the menu, the Forth system is available, type “words4” to see a list of all the Words (subroutines) available in the binary.

Interesting Words

- gpioa.
- rcc.
- free
- nvic.
- 2 2 + .
- 200 200 * .
- 0 207 0 305 f/ f.
- .” Hello World” cr
- green-on
- green-off

Development History

Developing this Forth bootable binary took a few days as I attempted to learn about the TSC peripheral, understand the STMicro TSC configuration ‘C’ code example and port it to Forth whilst testing on a STMicro F0 Discovery board.

Whilst very helpful, the ‘code example’ is incomplete because it doesn’t contain enough information to actually build a working prototype. This may explain why there are so few projects using the STM32 TSC on the Internet?

STMicro C TSC code example

Copied from the “STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM® - based 32-bit MCUs Reference Manual RM0091” dated January 2017.

```
A.18.1  TSC configuration code example

        /* Configure TCS */
        /* With a charge transfer around 2.5 µs */
        /* (1) Select fPGCLK = fHCLK/32,
          Set pulse high = 2xtPGCLK,Master
          Set pulse low = 2xtPGCLK
          Set Max count value = 16383 pulses
          Enable TSC */
        /* (2) Disable hysteresis */
        /* (3) Enable end of acquisition IT */
        /* (4) Sampling enabled, G2IO4 */
        /* (5) Channel enabled, G2IO3 */
        /* (6) Enable group, G2 */
        TSC->CR = TSC_CR_PGPSC_2 | TSC_CR_PGPSC_0 | TSC_CR_CTPH_0 |
TSC_CR_CTPL_0 | TSC_CR_MCV_2 | TSC_CR_MCV_1 | TSC_CR_TSCE; /* (1) */
        TSC->IOHCR &= (uint32_t)(~(TSC_IOHCR_G2_IO4 | TSC_IOHCR_G2_IO3));
/* (2) */
        TSC->IER = TSC_IER_EOAIE; /* (3) */
        TSC->IOSCR = TSC_IOSCR_G2_IO4; /* (4) */
        TSC->IOCCR = TSC_IOCCR_G2_IO3; /* (5) */
        TSC->IOGCSR |= TSC_IOGCSR_G2E; /* (5) */

A.18.2 TSC interrupt code example

        /* End of acquisition flag */
        if ((TSC->ISR & TSC_ISR_EOAF) == TSC_ISR_EOAF)
        {

           TSC->ICR = TSC_ICR_EOAIC; /* Clear flag */
           AcquisitionValue = TSC->IOGXCR[1]; /* Get G2 counter value */
        }
```

What’s Missing From The C Code Example?

1. GPIO Alternate Function selection code. Touch requires a minimum of two GPIO pins to be useful.
2. Starting an ‘acquisition’ by setting the ‘START’ BIT (1) of TSC_CR. Without this, nothing happens!

A more recent STMicro ‘Touch Sensor’ example I found was a series of GUI window pictures using Eclipse, HAL and STM32cube etc. There was no real code in this article, is this the dumbed-down future of embedded design?

Forth Code

This code excerpt from the touch demo binary shows the TSC peripheral config and interrupt code.

```
8 bit constant TSC-INTERRUPT  \ bitposn 8 settable TSC  Touch sensing
interrupt
0 variable task1-time
0 variable task1b-flag
6000 variable g2-counter
0 variable touched-flag
2500 variable sensitivity

\ ------------------------------------- Config Words --------------------
----------------------- \
: RCC_AHBENR_TSCEN ( -- x addr ) 24 bit RCC_AHBENR ; \ RCC_AHBENR_TSCEN,
Touch sensing controller clock  enable
: NVIC_ISER_SETENA ( %bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb -- x addr )
NVIC_ISER ; \ NVIC_ISER_SETENA, SETENA
: GPIOA_AFRL_AFRL7   ( %bbbb -- x addr ) 28 lshift GPIOA_AFRL ; \
GPIOA_AFRL_AFRL7, Alternate function selection
: GPIOA_AFRL_AFRL6   ( %bbbb -- x addr ) 24 lshift GPIOA_AFRL ; \
GPIOA_AFRL_AFRL6, Alternate function selection
: GPIOA_MODER_MODER7 ( %bb -- x addr ) 14 lshift GPIOA_MODER ; \
GPIOA_MODER_MODER7
: GPIOA_MODER_MODER6 ( %bb -- x addr ) 12 lshift GPIOA_MODER ; \
GPIOA_MODER_MODER6,
: TSC_CR_TSCE  ( -- 1 addr ) 0 bit TSC_CR ; \ TSC_CR_TSCE, Touch sensing
controller enable
: TSC_CR_START ( -- x addr ) 1 bit TSC_CR ; \ TSC_CR_START, Start a new
acquisition
: TSC_CR_MCV   ( %bbb -- x addr ) 5 lshift TSC_CR ; \ TSC_CR_MCV, Max
count value
: TSC_CR_PGPSC ( %bbb -- x addr ) 12 lshift TSC_CR ; \ TSC_CR_PGPSC,
pulse generator prescaler
: TSC_CR_CTPH  ( %bbbb -- x addr ) 28 lshift TSC_CR ; \ TSC_CR_CTPH,
Charge transfer pulse high
: TSC_CR_CTPL  ( %bbbb -- x addr ) 24 lshift TSC_CR ; \ TSC_CR_CTPL,
Charge transfer pulse low
: TSC_IOHCR_G2_IO4 ( -- x addr ) 7 bit TSC_IOHCR ; \ TSC_IOHCR_G2_IO4,
G2_IO4 Schmitt trigger hysteresis  mode
: TSC_IOHCR_G2_IO3 ( -- x addr ) 6 bit TSC_IOHCR ; \ TSC_IOHCR_G2_IO3,
G2_IO3 Schmitt trigger hysteresis  mode
: TSC_IER_EOAIE    ( -- 1 addr ) 0 bit TSC_IER ; \ TSC_IER_EOAIE, End of
acquisition interrupt  enable
: TSC_IOSCR_G2_IO4 ( -- x addr ) 7 bit TSC_IOSCR ; \ TSC_IOSCR_G2_IO4,
G2_IO4 sampling mode
: TSC_IOCCR_G2_IO3 ( -- x addr ) 6 bit TSC_IOCCR ; \ TSC_IOCCR_G2_IO3,
G2_IO3 channel mode
: TSC_IOGCSR_G2E   ( -- x addr ) 1 bit TSC_IOGCSR ; \ TSC_IOGCSR_G2E,
Analog I/O group 1 enable
: TSC_ICR_EOAIC    ( -- 1 addr ) 0 bit TSC_ICR ; \ TSC_ICR_EOAIC, End of
acquisition interrupt clear
: TSC_IOG2CR_CNT?  ( %bbbbbbbbbbbbbb -- x ) TSC_IOG2CR @ ; \
TSC_IOG2CR_CNT, Counter value
\ ------------------------------------- Config Words --------------------
---------------------- \

: tsc-config ( -- ) \ Init and configure the TSC peripheral ready for use
  RCC_AHBENR_TSCEN bis!

  \ GPIO: Alternate Function Mode: PA6 & PA7
  AF GPIOA_MODER_MODER7 bis!
  AF GPIOA_MODER_MODER6 bis!

  \ GPIO: AF3: PA6 & PA7
  AF3 GPIOA_AFRL_AFRL6 bis!  \ PA6 = TSC_G2_IO3
  AF3 GPIOA_AFRL_AFRL7 bis!  \ PA7 = TSC_G2_IO4

  \ 1. CONFIGURE TCS With a charge transfer around 2.5 µs
  %101 TSC_CR_PGPSC bis!  \ a. Select fPGCLK = fHCLK/32, 101: fHCLK /32
    %1 TSC_CR_CTPH bis!   \ b. Set pulse high = 2xtPGCLK,Master. CTPH:
0001: 2x tPGCLK
    %1 TSC_CR_CTPL bis!   \ c. Set pulse low = 2xtPGCLK. CTPL: 0001: 2x
tPGCLK
  %110 TSC_CR_MCV  bis!   \ d. Set Max count value = 16383 pulses. 110:
16383
       TSC_CR_TSCE bis!   \ e. Enable TSC. TSC_CR_TSCE, Touch sensing
controller enable

  \ 2. DISABLE HYSTERESIS 0: Schmitt trigger hysteresis disabled
       TSC_IOHCR_G2_IO4 bic! \ disable G2_IO4 Schmitt trigger hysteresis
       TSC_IOHCR_G2_IO3 bic! \ disable G2_IO3 Schmitt trigger hysteresis

  \ 3. ENABLE END OF ACQUISITION INTERRUPT
       TSC_IER_EOAIE bis! \  End of acquisition interrupt enable

  \ 4. SAMPLING ENABLED, G2IO4. 1: G2_IO4 used as sampling capacitor
  \ These bits are set and cleared by software to configure the G2_IO4 as
a sampling capacitor
  \ I/O. Only one I/O per analog I/O group must be defined as sampling
capacitor.
       TSC_IOSCR_G2_IO4 bis! \ G2_IO4 = sampling mode

  \ 5. CHANNEL ENABLED, G2IO3. 1: G2_IO3 used as channel
       TSC_IOCCR_G2_IO3 bis! \ G2_IO3 = channel mode

  \ 6. ENABLE GROUP, G2.
       TSC_IOGCSR_G2E bis!   \ TSC_IOGCSR_G2E = Analog I/O group
;
```

```
: g2-counter? ( -- x ) g2-counter @ ; \ This is the value of the sensor
read, high values around 5000 mean no touch

: acquire ( --  )      \ TSC_CR_START, Start a new acquisition
   TSC_CR_START bis!   \ start acquisition, it finishes with a interrupt
by the irq-tsc
;                      \ which updates g2-counter within about 5
milliseconds.

: tsc-handler ( -- )   \ Interrupt occurs because voltage across CS has
reached the threshold
  TSC_ICR_EOAIC bis!   \ TSC_ICR_EOAIC,  Clear "End of acquisition
interrupt flag"
  TSC_IOG2CR_CNT?  g2-counter ! \ count = number of charge transfer
cycles generated to complete acquisition
;

: tsc-init ( -- )
  ['] tsc-handler irq-tsc !         \ tie handler to interrupt
  TSC-INTERRUPT NVIC_ISER_SETENA !  \ enable TSC interrupt
;
```